НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

На правах рукопису
УДК 004.4

До захисту допущено
В. о. завідувача кафедри ММСА
О.Л.Тимощук
«___» _____ 2019 р.

Магістерська дисертація

на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки
спеціалізація Системи штучного інтелекту
на тему: «Інтелектуальна система кластеризації помилок
як складова автоматизації тестування»

Виконала:
студентка ІІ курсу, групи КА-з82мп
Глушко Софія Михайлівна _____

Керівник: доцент кафедри ММСА
к.т.н, доц. Дідковська М. В. _____

Рецензент:

_____

Засвідчую, що у цій магістерській дисертації
немає запозичень з праць інших авторів
без відповідних посилань
Студент _____

Київ
2019

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
ІМЕНІ ІГОРЯ СІКОРСЬКОГО»
ІНСТИТУТ ПРИКЛАДНОГО СИСТЕМНОГО АНАЛІЗУ
КАФЕДРА МАТЕМАТИЧНИХ МЕТОДІВ СИСТЕМНОГО АНАЛІЗУ

Рівень вищої освіти — другий (магістерський)
Спеціальність — 122 «Комп'ютерні науки»

ЗАТВЕРДЖУЮ
В. о. завідувача кафедри ММСА
О. Л. Тимощук
«___» _____ 2019 р.

ЗАВДАННЯ
на магістерську дисертацію студентці Глушко Софії Михайлівні

1. Тема дисертації: «Інтелектуальна система кластеризації помилок як складова автоматизації тестування», науковий керівник дисертації Дідковська Марина Віталіївна, к.т.н., доцент, затверджені наказом по університету від 05 листопада 2019 р. № 3825-с.
2. Термін подання студентом дисертації: 13 грудня 2019 р.
3. Об'єкт дослідження: інтелектуальний аналіз результатів тестування.
4. Предмет дослідження: система кластеризації повідомлень про помилки тестування.
5. Перелік завдань, які потрібно розробити:
1) дослідити сучасний стан, наявність та придатність систем інтелектуального аналізу результатів тестівання;
2) провести аналіз задачі кластеризації текстових повідомлень про помилки тестування;
3) дослідити наявні моделі векторизації тексту;
4) розробити власний алгоритм векторизації тексту на базі існуючих;
5) дослідити наявні моделі кластеризації даних, їхні особливості та недоліки;
6) розробити власний алгоритм кластеризації даних на базі існуючих;
7) розробити архітектуру програмного продукту;
8) реалізувати описані алгоритми у вигляді комп'ютерної програми;
9) розробити стартап-проект виведення на ринок результатів дослідження;
10) розробити висновки та рекомендації за результатами дослідження.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу:
1) діаграми архітектури програмного продукту (рисунки);
2) результати роботи програмного продукту (рисунки);
3) таблиці у розділі стартап-проекту.

7. Дата видачі завдання: 05 вересня 2019 р.

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації
1.	Концептуальний вступ дисертації. Формулювання об’єкта, предмета, цілі, завдань, новизни, практичної значущості результатів.	05.09.2019—15.09.2019
2.	Перший розділ. Аналіз задачі кластеризації помилок. Огляд наявних підходів до інтелектуального аналізу помилок тестування. Характеристика об’єкта. Формалізація постановки задачі.	16.09.2019—27.09.2019
3.	Огляд літературно-інформаційних джерел. Понятійно-категоріальний апарат. Аналіз існуючих методів кластеризації даних.	03.10.2019—25.10.2019
4.	Стартап-проект.	26.10.2019—29.10.2019
5.	Огляд літературно-інформаційних джерел. Понятійно-категоріальний апарат. Аналіз існуючих методів векторизації тексту.	01.11.2019—05.11.2019
6.	Оформлення другого та третього розділів.	06.11.2019—09.11.2019
7.	Імплементація отриманих результатів у програмний продукт. Тестування програми. Оформлення четвертого розділу.	10.11.2019—26.11.2019
8.	Порівняльний аналіз отриманих результатів роботи програмного продукту. Оформлення п’ятого розділу.	27.11.2019—02.12.2019
9.	Концептуальні висновки. Перспективи розвитку отриманих рішень	03.12.2019—05.12.2019

Студент С. М. Глушко

Науковий керівник дисертації М. В. Дідковська

РЕФЕРАТ

Магістерська дисертація: 125 с., 26 табл., 16 рис., 1 дод., 29 джерел.

КЛАСТЕРИЗАЦІЯ, ВЕКТОРИЗАЦІЯ ТЕКСТУ, ТЕСТУВАННЯ, ПОМИЛКА, ІНТЕЛЕКТУАЛЬНИЙ АНАЛІЗ ДАНИХ.

Об’єктом дослідження є інтелектуальний аналіз результатів тестування.

Предметом дослідження є системи кластеризації помилок тестування.

Мета роботи – розробити алгоритм для кластеризації результатів тестування, які представлені повідомленнями про помилки тестування.

Методом дослідження є методи векторизації тексту та методи кластеризації даних.

Магістерська дисертація складається із шести розділів. У першому розділі проаналізовано та формалізовано задачу кластеризації помилок тестування. У другому розділі розглядаються та аналізуються наявні підходи до векторизації текстів. У третьому розділі досліджуються та аналізуються існуючі підходи до кластеризації даних. Четвертий розділ описує алгоритм векторизації та кластеризації текстових повідомлень про помилки тестування, архітектуру розробленої програми. П’ятий розділ містить аналіз результатів роботи алгоритмів. У шостому розділі наведені опис, ідея та маркетингова стратегія стартап-проекту.

ABSTRACT

Masters' thesis: 125 pages, 26 tables, 16 figures, 1 appendix, 29 sources.

CLUSTERISATION, TEXT VECTORISATION, TESTING, ERROR, DATA MINING.

The object of this research is test results mining.

The subject of this research is test errors clustering system.

The purpose of this research is to develop an algorithm to clasterize the test results, which are presented in the form of test error messages.

The masters' thesis consists of six sections. In the first section the problem of test errors clustering is analyzed and formalized. In the second section existing text vectorization approaches are analyzed and compared. In the third section existing clustering approaches are investigated and analyzed. The fourth section describes the test error text messages vectorizing and clustering algorithm and developed program architecture. The fifth section contains algorithms execution results analysis. The sixth section explains the idea and the marketing strategy of a start-up project.

ЗМІСТ

ВСТУП

На ринку України сфера інформаційних технологій у 2019 році зайняла третє місце за об’ємом експорту послуг з долею 20% від усього експорту сфери послуг. У країні працює більше 1600 компаній, що надають послуги з розробки програмного забезпечення. [1] Однією із найбільш важливих складових життєвого циклу розробки програмного забезпечення є тестування. Тестування програмного забезпечення – це сфера, яка вважається чи не найважливішою у сучасному світі технологій, що швидко змінюються. Тестування відбувається на різних етапах розробки заради забезпечення якості (quality assurance) продукту, що доставляється.

Аналіз результатів виконання тестів, а саме аналіз падінь (неочікуваних результатів виконання тесту), – одна з основних складових забезпечення якості програмних продуктів. Проте ця робота часто є рутинною, повторюваною, займає велику кількість часу тестувальників. При достатньо великих кількостях автоматизованих тестів, наприклад, більше 1000, кількість результатів із помилками теж може бути великою. При регулярному виконанні тестів – наприклад, щоночі – командам тестувальників доводиться створювати окрему посаду з єдиним обов’язком – щоденний аналіз результатів виконання тестів. Дана робота може бути автоматизована, а звільнений час команди тестувальників – витрачений на розробку нових тестів та виконання інших обов’язків.

Метою даної магістерської дисертації є зменшення часу та ресурсів, що витрачаються на аналіз результатів виконання тестування, за рахунок використання інтелектуальної системи кластеризації помилок, яка забезпечить групування схожих результатів виконання

тестів, виявлення як вже відомих, так і нових причин неуспішності тестів.

Кластеризація помилок дозволить швидко виявляти закономірності, знаходити одразу всю групу тестів, які падають через аналогічну помилку. Це дозволить тестувальникам аналізувати результати регулярного виконання тестів з урахуванням подібності між результатами, що потрапили до одного кластеру, замість перебору усіх виявлених помилок.

Завданням даної магістерської дисертації є аналіз наявних алгоритмів кластеризації даних та, на основі проведеного аналізу, розробка системи кластеризації помилок тестування, представлених у вигляді текстових повідомлень.

Об’єктом даного дослідження є інтелектуальні системи аналізу результатів тестування.

Предметом даного дослідження є система кластеризації помилок тестування.

Новизна даного дослідження полягає у вперше запропонованому алгоритмі кластеризації помилок у системах автоматизованого тестування на основі алгоритму агломеративного ієрархічного поділу.

Практична цінність даного дослідження полягає у розробленій системі кластеризації помилок, дозволила знизити час аналізу результатів виконання автоматизованого тестування на 30%.

Дана магістерська дисертація складається із шести розділів. У першому розділі проведено аналіз задачі кластеризації помилок, описано актуальність задачі кластеризації помилок, оглянуто наявні засоби інтелектуального аналізу результатів тестування, формалізовано постановку задачі. У другому розділі проаналізовано наявні методи векторизації тексту, проведено порівняльний аналіз розглянутих методів, запропонований модифікований метод векторизації тексту. У

третьому розділі розглянуто наявні методи кластеризації даних, проведено порівняльний аналіз розглянутих методів, запропоновано модифікований алгоритм кластеризації даних. У четвертому розділі обґрунтовано вибір засобів розробки та архітектури програмного продукту, наведено керівництво користувача. У п’ятому розділі проаналізовано результати роботи програмного забезпечення. У шостому розділі проведено технічний аудит ідеї продукту, сформовано ринкову стратегію та маркетингову програму стартап-проекту.

РОЗДІЛ 1 АНАЛІЗ ЗАДАЧІ КЛАСТЕРИЗАЦІЇ ПОМИЛОК

У даному розділі магістерської дисертації буде розглянуто актуальність задачі кластеризації помилок тестування, розглянуто наявні підходи до інтелектуального аналізу результатів тестування, виявлено їхні переваги та недоліки, формалізовано постановку задачі магістерської дисертації.

1.1 Актуальність задачі кластеризації помилок

Тестування виступає важливим етапом розробки програмного забезпечення. На основі результатів тестування роблять висновки про необхідність подальшої розробки, необхідність виправлення дефектів продукту, можливість випуску програмного продукту для кінцевих користувачів. Для прийняття будь-яких рішень на основі результатів тестування необхідно провести їх аналіз. Більшість реальних проектів мають набори тестів, кількість яких визначається тисячами, і відповідно аналіз результатів виконання такої кількості тестів вручну займає лінійно співвідносну кількість часу. В умовах continuous delivery, тобто безперервної поставки продукту, такий аналіз потрібно виконувати щодня.

За описаних вище умов, автоматичний аналіз результатів тестування дозволить скоротити час, що витрачається тестувальниками, а вивільнений – застосувати для виконання інших обов’язків тестової команди.

1.2 Особливості предметної області

Результати виконання тестів можуть подаватися у вигляді xml- або json-колекцій елементів, які містять дані про кожен тест, що був виконаний у даному наборі, його назву, мітки, інформацію про час його виконання, його результат. У разі, якщо результат виконання тесту неуспішний, колекція також містить повідомлення про помилку тесту у вигляді рядка тексту. Саме такі повідомлення і є вихідними даними для інтелектуального аналізу. Оскільки алгоритми кластеризації працюють лише з числовими вхідними даними, повідомлення про помилки тестування необхідно попередньо трансформувати у числові датасети, тобто набори числових векторів, за допомогою методів векторизації тексту. Таким чином вхідними даними для кластеризації є багатовимірні числові вектори, а вихідними – список отриманих кластерів, який містить тексти відповідних помилок та назви відповідних даним помилкам тестів.

1.3 Наявні підходи до агрегації та аналізу результатів виконання тестів

Серед наявних засобів, що можуть бути використані у роботі тестувальників задля агрегації, аналізу та візуалізації результатів тестування, у тому числі помилок, можна виділити такі:

- SerilogAmazonKinesis – програмний пакет, який інтегрується з модулем логування тестового фреймворку. Фактично, цей програмний продукт забезпечує якісне логування в одну зі спеціалізованих систем,

наприклад Splunk, проте весь функціонал з аналізу логів надається виключно системами для збору логів. Вищезгаданий Splunk – це система для збору, збереження та обробки машинних даних, або логів. Обробка відбувається за допомогою SPL запитів (спеціальна мова), за допомогою яких можна будувати різноманітні вибірки та таблиці, сортувати, фільтрувати, агрегувати дані, будувати звіти, створювати обчислювані поля, звертатися як до внутрішніх, так до зовнішніх довідників, створювати візуальні репрезентації інформації, а також створювати сповіщення та відкладені події. Незважаючи на повноту функціоналу система Splunk не виконує інтелектуального аналізу, вона забезпечує лише вибірки з логів. Якщо розглядати вартість, Splunk дорогий, а безкоштовна версія системи з обмеженням на індексацію логів до 500 Мб на день є істотною перешкодою, що нівелює причини звернення до систем машинного аналізу, а також потребує окремих налаштувань і підтримки.

- ReportPortal – це програмний продукт з відкритим кодом, створена розробниками EPAM та OSS-спільноти. Її використання дозволяє збирати в одному місці документи та результати тестування різних проектів. Ця система забезпечує вирішення таких завдань як перегляд тестових сценаріїв зі всіма пов’язаними даними, з логами, скріншотами та двійковими даними, в одному місці та пов’язування певних тестових сценаріїв зі знайденими багами (дефектами програмного забезпечення), проблемами автоматизації та проблемами системи. Також функціонал системи ReportPotal включає такі можливості:

- робота з результатами автоматизації різних проектів в одному місці;

- централізована агрегація логів, звітів та медіа даних в реальному часі;

- спільна робота над аналізом результатів тестування;
- присвоєння кожній помилці тестування посилання на відомий баг програмного продукту, проблеми автоматизації, системної проблеми чи будь-якого іншого типу помилки, визначеного та описаного самостійно;
- збереження історії проходження тестового сценарію;
- надсилання деталей помилки напряму до системи відстежування помилок;
- візуалізація даних;
- інтеграція з різноманітними тестовими фреймворком із різними мовами програмування;
- миттєве відображення результатів, навіть у процесі тестування;
- автоматичний аналіз результатів тестування, який включає категоризацію посилок за попередніми результатами, такими як прив'язка результатів тестування до багів системи, автоматизаційних чи системних проблем і т. д.

Таким чином, загалом це чудова система для використання в роботі тестувальників, вона дозволяє зберігати і візуалізувати велику кількість результатів, незважаючи на складність налаштування. Проте аналіз результатів працює неякісно, за відгуками багатьох користувачів – видає некоректний, неадекватний та неактуальний результат, і до того ж не виявляє закономірностей в нових помилках. Авто-аналіз системи ReportPortal побудований на ElasticSearch і в цілому нагадує фільтр для системи Splunk. Завдяки тому, що до ReportPortal можна підключити власну систему для аналізу результатів, його можна використати як графічну оболонку для розробленої у даній дисертації системи кластеризації результатів тестування.

Більшість із інших наявних засобів для агрегації результатів тестування (SerilogAmazonKinesis, AventStack.ExtentReports, Specrunner) забезпечують лише візуалізацію і не надають жодних можливостей для інтелектуального аналізу даних.

1.4 Формалізація постановки задачі

Метою даної магістерської дисертації є розв'язок часткового випадку задачі кластеризації даних, а саме кластеризації помилок тестування.

Розроблена в результаті даного дослідження система повинна забезпечувати виявлення закономірностей серед негативних результатів тестування – об'єднання результатів у кластери, де всі тести падають з однакової причини.

Таким чином необхідно створити систему, що здатна кластеризувати отриманий набір результатів виконання тестів, поданий у загальновживаному форматі, наприклад, xml, на основі їхніх помилок, а саме повідомлень про помилки. У подальших дослідженнях можна розробити інтеграцію модулю із системою ReportPortal.

Для реалізації необхідно:

- дослідити відомі методи векторизації тексту;
- векторизувати повідомлення про помилки за допомогою певного засобу пре-процесингу тексту для підготовки до застосування алгоритмів інтелектуального аналізу даних;
- проаналізувати відомі алгоритми кластеризації даних для реалізації інтелектуального аналізу даних;

- сформувати критерії оцінки відомих алгоритмів кластеризації даних;
- вибрати найкращий алгоритм кластеризації даних для побудови системи кластеризації помилок тестування;
- розробити архітектуру та програмний код інтелектуальної системи кластеризації помилок;
- проаналізувати отримані результати роботи програмного продукту;
- сформулювати висновки.

Висновок до розділу 1

У даному розділі описано актуальність задачі, що досліджується у даній магістерській дисертації, розглянуто та проаналізовано можливості наявних засобів, що можуть бути використані для агрегації та аналізу результатів тестування. Обрано систему ReportPortal, яка, за специфікацією, виконує класифікацію результатів тестування, проте за відгуками користувачів не надає адекватних результатів роботи. Проаналізовано задачу кластеризації даних, векторизації текстових повідомлень. Сформульовано постановку задачі даної магістерської дисертації.

РОЗДІЛ 2 МАТЕМАТИЧНІ ОСНОВИ ЗАДАЧІ ВЕКТОРИЗАЦІЇ ТЕКСТУ

У даному розділі розглянуто та проаналізовано відомі методи векторизації тексту.

Проблема моделювання тексту полягає в тому, що він безладний для комп'ютерних систем, а техніки машинного навчання приймають на вхід чітко визначені параметри. Алгоритми не працюють із необробленим текстом.

Для того, щоб використовувати дані про помилки тестування в алгоритмах кластеризації, необхідно представити їх у вигляді числових датасетів, або векторів.

2.1 Аналіз наявних методів векторизації тексту

Для векторизації тексту можна застосувати декілька різних методів, таких як Bag of words, TF-IDF та Word2Vec. [3]

При цьому необроблені дані можуть містити цифрові значення, пунктуацію, спеціальні символи та інші входження, які не мають прямого стосунку до змісту, тому перед тим, як застосовувати алгоритми векторизації тексту, його необхідно підготувати, очистити і нормалізувати, застосувавши Text Preprocessing. [4]

Першим кроком є видалення шумів даних. Такими можуть бути xml- та html-теги, заголовок та закінчення, розмітка. Коли такі дані не несуть ніякого сенсу, їх обов’язково потрібно видалити, наприклад, за допомогою засобів регулярних виразів.

Наступним кроком очищення даних є токенізація, або сегментація тексту, чи лексичний аналіз. Власне, цей процес полягає у розбитті рядка даних на слова чи символи, створення масиву слів.

Результат сегментації даних може містити знаки пунктуації, стоп-слова, які нерелевантні у даній ситуації (наприклад, займенники), великі та малі літери, тому не може вважатися остаточним. Наступним кроком є нормалізація отриманих даних, яка забезпечується стемінгом або лематизацією. Стемінг зазвичай полягає у грубому евристичному процесі, який відрізає закінчення слів, а також часто і похідні афікси. Під лематизацією, як правило, мається на увазі лише видалення флективних форм з метою повернення до словникової форми слова, яка називається лемою. Ці два методи найчастіше відрізняються тим, що стемінг згортає похідні форми слова до однієї, тоді як лематизація руйнує лише флективні форми леми. [5] З точки зору граматики, ми використовуємо різні форми слів, проте при пошуку зв’язків доречним є шукати лише одну форму для усіх спільнокореневих слів. Метою як стемінгу, так і лематизації є зменшення кількості флективних, похідних та споріднених форм слова до однієї основної форми. Лінгвістична обробка стемінгом чи лематизацією зазвичай виконується додатковим компонентом чи плагіном, серед яких є багато рішень із відкритим кодом.

2.1.1 Bag of words

Bag of words, або мішок слів, – базова модель, що використовується в обробці природної мови. Вона характеризується наявністю словника відомих слів та мірою присутності даних слів у

тесті. Таку назву дана модель отримала через те, що відкидає будь-який порядок слів у документі і вказує лише на присутність чи відсутність певного слова. У даному методі робиться припущення, що документи однакові, якщо вони містять однакові слова. Ускладнення алгоритму відбувається за рахунок визначення принципу складання словника та принципу підрахунку входжень даних слів. [6]

Наприклад, нехай наступні речення (2.1) – окремі документи:

“There used to be Stone Age”,
“There used to be Bronze Age”,
“There used to be Iron Age”,
“There was Age of Revolution”,
“Now it is Digital Age” (2.1)

Список усіх слів, що зустрічаються хоча б в одному документі наведено у 2.2.

“There”,”was”,”to”,”be”,”used”,”Stone”,”Bronze,”Iron”,
”Revolution”,”Digital”,”Age”,”of”,”Now”,”it”,”is” (2.2)

Слова конвертуються у вектори на підставі кількості їхніх входжень до певного документу, наприклад, речення “There used to be Stone Age” трансформується в наступне (2.3):

$$
\begin{aligned}
&\textit{“There”} = 1,\\
&\textit{”was”} = 0,\\
&\textit{”to”} = 1,\\
&\textit{”be”} = 1,\\
&\textit{”used”} = 1,\\
&\textit{”Stone”} = 1,\\
&\textit{”Bronze”} = 0,\\
&\textit{“Iron”} = 0,\\
&\textit{”Revolution”} = 0,\\
&\textit{”Digital”} = 0,\\
&\textit{”Age”} = 1,\\
&\textit{”of”} = 0,\\
&\textit{”Now”} = 0,\\
&\textit{”it”} = 0,\\
&\textit{”is”} = 0
\end{aligned}
\tag{2.3}
$$

Таким чином речення трансформуються у вектори (2.4).

$$
\begin{aligned}
&\textit{“There used to be bronze age”} = [1,0,1,1,1,0,1,0,0,0,1,0,0,0,0],\\
&\textit{“There used to be iron age”} = [1,0,1,1,1,0,0,1,0,0,1,0,0,0,0],\\
&\textit{“There was age of revolution”} = [1,1,0,0,0,0,0,0,1,0,1,1,0,0,0],\\
&\textit{“Now its digital Age”} = [0,0,0,0,0,0,0,0,0,1,1,0,1,1,1]
\end{aligned}
\tag{2.4}
$$

Описаний вище підхід називається уніграмою, оскільки бере до уваги по одному слову за раз. Біграмою називається підхід, яких оцінює сполучення із двох слів, наприклад, There used, Used to, to be, і так далі. [3]

Отримані вектори можуть бути об’єднані в розріджену матрицю, як показано на рисунку 2.1.

```
array([[1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0],
       [1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
       [1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0],
       [1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1],
       [1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0]], dtype=int64)
```

Рисунок 2.1 – Матриця алгоритму Bag of words

Незважаючи на простоту підходу, він має декілька обмежень. Словник потребує продуманого дизайну, перш за все для роботи з його розміром. Відкидання порядку слів також відкидає і їхнє семантичне значення, зв’язок між словами, контекст. Проблема підрахунку кількості появ слова в документах полягає у тому, що дуже часті слова починають домінувати, в той час як вони насправді можуть не нести ніякого суттєвого змісту.

2.1.2 TF-IDF

TF-IDF розшифровується як Term Frequency – Inverse Document Frequency, тобто даний метод в першу чергу вказує на важливість слова в усьому датасеті. TF-IDF складається з двох основних концептів – Term Frequency (TF), або частота виразу, та Inverse Document Frequency (IDF), або обернена частота документу.

Term Frequency визначається як частота появи слова в усьому наборі документів. Оскільки усі речення різної довжини, виникає ймовірність появи одного і того ж слова більше разів у довшому реченні. Тому Term Frequency для кожного виразу в документі визначається формулою 2.5.

$$TF_{term,document} = \\ = \frac{Number\ of\ times\ term\ appears\ in\ the\ document}{Total\ number\ of\ words\ in\ the\ document} \quad (2.5)$$

Оскільки частота появи деяких дуже поширених слів, наприклад англійського артикля «the», може бути значно більшою у порівнянні із рештою слів, застосовується логарифмування значень за формулою 2.6.

$$W_{t,d} = \begin{cases} 1 + \log_{10} TF_{t,d}, & TF_{t,d} > 0, \\ 0, & TF_{t,d} \leq 0 \end{cases} \quad (2.6)$$

Inverse Document Frequency базується на припущенні, що менш часто вживані слова є більш важливими для тексту. [3] Inverse Document Frequency для кожного слова визначається формулою 2.7, в якій також застосовується логарифмування.

$$IDF_{term} = \log_{10} \frac{Number\ of\ documents}{Number\ of\ documents\ the\ term\ appears\ in} \quad (2.7)$$

Таким чином, якщо слово зустрічається у всіх документах, даний показник для нього буде дорівнювати нулю.

Показник TF-IDF для кожного слова у кожному документі – це, власне, добуток двох описаних вище величин (формула 2.8).

$$\mathrm{TF} - \mathrm{IDF}_{t,d} = TF_{t,d} * IDF_t \quad (2.8)$$

Перевагою даного алгоритму є вирішення проблеми стоп-слів, наприклад, артиклів, які, скоріше за все, з’являються у всіх документах з набору, а тому отримують найнижчі оцінки. TF–IDF широко

використовується як покращення Bag of words і є оптимальним для розв’язку базових задач аналізу текстів. [5]

2.1.3 Word2Vec

Алгоритм Word2Vec – компонент бібліотеки TensorFlow від компанії Google. Це техніка глибинного навчання, побудована на основі двошарової нейронної мережі. Вона приймає на вхід великі масиви даних і трансформує їх у векторні простори. Word2Vec попередньо навчений на датасетах компанії Google. [3]

Найпростіший опис роботи даного алгоритму полягає в тому, що він намагається розмістити усі слова у багатовимірному просторі так, щоб їхнє положення визначалось їхнім значенням. Це означає, що слова зі схожим значенням будуть розташовані поблизу. Слова із семантичними зв’язками будуть розташовані ближче одне до одного, ніж слова без них.

Цікаво, що проміжки і відстані між словами у такій візуалізації також мають значення, наприклад, якщо взяти слово “king” і побудувати з нього вектор такий самий, як між словами “man” і “woman”, то на кінці даного вектору буде знаходитись слово “queen”.

Потенційними сферами застосування даного алгоритму є представлення масивів знать, машинний переклад, відповіді на запитання, розмовні системи та багато інших.

У зв’язку з даним підходом виникло багато теоретичних запитань та обмежень, найбільший з яких полягає в тому, що контекст – це значна частина мови і будь-яка модель типу «мішок слів» може бути обмеженою у своєму застосуванні. Більш широке занепокоєння

викликала думка, що машинне навчання може бути лише настільки якісним, наскільки якісними були вихідні дані для нього. Якщо говорити більш детально з точки зору розуміння мови, то будь-які упередження, закладені до масиву вихідних даних, будуть відображені і в моделі, отриманій за допомогою машинного навчання. Тобто якщо тренувальна вибірка містить сексизм, то машина сформує сексистські асоціації. [7]

2.2 Порівняльний аналіз розглянутих методів векторизації тексту

Перший метод, Bag of words, дуже простий в реалізації та використанні, проте дає досить неточні результати, оскільки майже всі повідомлення про помилки містять вираз «System.Exception» і т. д. Підвищити його ефективність можна за допомогою модифікації списку стоп-слів, куди додати, окрім загальновживаних займенників, прийменників і коротких дієслів, також поширені у технічних повідомленнях терми. Таким чином, при правильному складанні словника, в отриманих векторах залишаться лише ті терми, які визначають суть помилки.

Другий метод, TF-IDF, на перший погляд, здається більш підходящим, оскільки він одразу враховує та відкидає загальновживані терми, проте він може працювати не зовсім коректно у випадку, коли усі вхідні значення однакові. З іншого боку, враховуючи особливості даної задачі, у випадку однакових повідомлень даний метод видасть як результат набір нульових векторів, які є однаковими, як і повідомлення, що повністю відображає кореляцію між ними. З урахуванням усіх особливостей, такий результат є задовільним. З точки зору обчислень

даний метод вимагає дещо більших потужностей, ніж попередній, проте краще нормалізує вектори, що забезпечує нижчу складність даних для подальших обчислень.

Водночас модифікацію словника, а саме використання спеціалізованого набору стоп-слів, можна застосовувати як для першого методу, так і для другого.

Третій метод, Word2vec, є достатньо потужним для пошуку закономірностей у семантичних зв'язках між словами, проте повідомлення про помилки у програмному забезпеченні зазвичай містять лише сухі дані про події. Незначні семантичні відхилення, які даний метод практично не розрізняє у числовому представленні, можуть бути визначальними відмінностями між двома помилками тестування. У зв'язку із цією особливістю третього з розглянутих методів, а також із його складністю обчислень, реалізації та необхідністю використання додаткових засобів, даний метод не застосовний для розв'язання поставлених задач.

Також, у зв'язку із необхідністю максимально можливого у даному контексті розрізнення семантичних форм, для попереднього очищення даних буде застосовуватися метод лематизації замість загальнопоширеного стемінгу.

2.3 Модифікація алгоритму векторизації тексту

Оскільки жоден із розглянутих методів повністю не підходить для розв'язання поставленої у даній магістерській дисертації задачі, запропоновано такі модифікації вищеописаних алгоритмів:

- модифікація словника шляхом вилучення загальновживаних в описі помилок тестування слів та фраз, наприклад, терму exception;
- заміна стемінгу на лематизацію задля очищення вхідних даних;
- застосування нормалізації отриманих числових векторів для зменшення складності даних.

Висновок до розділу 2

У даному розділі магістерської дисертації було проаналізовано різні підходи до розв’язку задачі векторизації тексту, виявлено їхні особливості. З розглянутих алгоритмів найкращим для реалізації у даній роботі виявився алгоритм TF-IDF з модифікаціями у створенні словника, які адаптують його до предметної області, а саме тестування і текстів помилок програмного забезпечення. Даний алгоритм достатній для формування адекватних векторів виходячи із складності даних, які йому передаватимуть.

Алгоритм Word2Vec базується на встановленні семантичних зв’язків, що для розв’язку даної задачі є надлишковим, тому було вирішено використовувати більш прості методи.

Після того, як дані стали векторами, до них можна застосовувати різні алгоритми, в тому числі функції порівняння. Тобто далі річ піде про власне класифікацію і кластеризацію масивів даних.

РОЗДІЛ 3 МАТЕМАТИЧНІ ОСНОВИ ЗАДАЧІ КЛАСТЕРИЗАЦІЇ ДАНИХ

У даному розділі магістерської дисертації розглянуто математичні основи задачі кластеризації даних та проведено теоретичний аналіз оглянутих методів кластеризації даних.

Задача кластерного аналізу є частковим випадком експерименту навчання без учителя. Навчання без вчителя – один зі способів машинного навчання, при вирішенні яких випробовувана система спонтанно навчається виконувати поставлене завдання, без втручання з боку експериментатора. Воно часто протиставляється навчанню з учителем, коли для кожного об'єкта, що навчається, примусово задається «правильна відповідь», і потрібно знайти залежність між стимулами та реакціями системи. [8]

При кластеризації вибірка об'єктів розбивається на підмножини, що не перетинаються (вони називаються кластерами), так, щоб кожен кластер складався зі схожих об'єктів, а об'єкти різних кластерів суттєво відрізнялись. Початкова інформація представляється у вигляді матриці відстаней.

Кластеризація може виконувати допоміжну роль при розв'язанні задач класифікації та регресії. Для цього треба спочатку розбити вибірку на кластери, потім до кожного кластера застосувати який-небудь простий метод, наприклад, наблизити цільову залежність константою. [9]

Оскільки поняття «кластеру» не може бути точно визначено, існує багато різних методів кластеризації. Використовують різні моделі кластерів, і для кожної з цих моделей можуть бути застосовані різні алгоритми. Поняття кластера, які отримуються у різних алгоритмах,

різняться властивостями. Розуміння цих «кластерних моделей» є ключовим для розуміння відмінностей між різними алгоритмами. Типовими кластерними моделями є:

- Моделі зв'язності. Наприклад, ієрархічна кластеризація або таксономія будуються на основі відстані між вузлами.
- Центроїдні моделі. Наприклад, метод K-середніх (K-means) представляє кожен кластер єдиним усередненим вектором.
- Статистичні моделі. Кластери будуються ґрунтуючись на статистичних розподілах. Таких як багатовимірний нормальний розподіл з допомогою EM-алгоритму.
- Моделі засновані на щільності. Наприклад, в DBSCAN кластери визначаються як зв'язані області відповідної щільності у просторі даних.
- Групові моделі. Деякі алгоритми не забезпечують вдосконалену модель для своїх результатів, а просто описують групування об'єктів.
- Графові моделі. Поняття кліки (така підмножина вершин, в якій кожна пара вершин з'єднана ребром) у графі слугує прототипом кластеру. Пом'якшення вимоги до повної зв'язності (тобто, частина ребер може бути відсутня) призводить до поняття відомого як квазі-кліка. Вони будуються алгоритмом HCS.
- Нейронні моделі. Найбільш відомою моделлю нейронної мережі з навчанням без учителя є нейронна мережа Кохонена. Ці моделі, як правило, можна охарактеризувати як схожі на одну або подібні якійсь з наведених вище моделей. [10]

3.1 Аналіз наявних методів кластеризації даних

У даному підрозділі розглянуто та проаналізовано такі методи та алгоритми кластеризації даних, як агломеративний (AHC), DBSCAN, HDBSCAN, HCS, EM-алгоритм, кластеризація методом к-середніх та нейронні мережі Кохонена.

3.1.1 Нейронні моделі

Штучні нейронні мережі, або конективістські системи — це обчислювальні системи, натхнені біологічними нейронними мережами, що складають мозок тварин. Такі системи навчаються задач (поступально покращують свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під задачу. Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як «кіт» і «не кіт», і використовуючи результати для ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів, наприклад, що вони мають хутро, хвости, вуса та котоподібні писки. Натомість, вони розвивають свій власний набір доречних характеристик з навчального матеріалу, який вони оброблюють.

Первинною метою підходу ШНМ було розв'язання задач таким же способом, як це робив би людський мозок. З часом увага зосередилася на відповідності лише певним розумовим здібностям, ведучи до відхилень від біології. ШНМ використовували в ряді різноманітних

задач, включно з комп'ютерним баченням, розпізнаванням мовлення, машинним перекладом, соціально-мережевим фільтруванням, грою в настільні та відеоігри, та медичним діагностуванням. [11]

Самі по собі системи нейронних мереж – занадто широке поняття для використання у кластеризації, тож далі буде розглянутий такий підклас нейронних мереж, як нейронні мережі Кохонена.

Нейронні мережі Кохонена — клас нейронних мереж, основним елементом яких є шар Кохонена. Шар Кохонена складається з адаптивних лінійних суматорів («лінійних формальних нейронів»). Як правило, вихідні сигнали шару Кохонена обробляються за правилом «переможець забирає все»: найбільший сигнал перетворюється в одиничний, решта перетворюються в нуль. [12]

За способами налаштування вхідних ваг суматорів і за завданнями, які розв’язують дані нейронні мережі, розрізняють багато різновидів мереж Кохонена [13]. Найбільш відомі із них:

- мережі векторного квантування сигналів [14], тісно пов'язані з найпростішим базовим алгоритмом кластерного аналізу (метод динамічних ядер або K-середніх);
- самоорганізаційні карти Кохонена (Self-Organising Maps, SOM);
- мережі векторного квантування, які вивчаються з учителем (Learning Vector Quantization). [15]

Карти Кохонена використовуються, в першу чергу, для візуалізації та початкового аналізу даних. Важливою відмінністю самоорганізаційних карт Кохонена є те, що у даному методі усі нейрони мережі впорядковані у певну двовимірну структуру, яка нагадує сітку, і в ході навчання зазвичай змінюється вага не лише нейрона-переможця, а і його сусідів. Завдяки цьому самоорганізаційні карти Кохонена можна

назвати методом проектування багатовимірного простору на простір із меншою розмірністю.

Навчання нейронної мережі Кохонена відбувається за таким алгоритмом:

- Вибір конфігурації сітки, кількості нейронів у мережі. Ініціалізація ваг W малими випадковими значеннями. Визначення початкового розміру області сусідства $\alpha(n)$ та коефіцієнта швидкості навчання $\sigma(n)$. Задання критерію зупинки алгоритму μ.
- Подання вектора вхідних даних x на вхід мережі.
- Порівняння x з усіма векторами вагових коефіцієнтів w_j та обрання такого вихідного вузла j, щоб відстань $d_j = \min_i(D)$. Відстань між вектором вагових коефіцієнтів найближчого нейрона, або нейрона-переможця, та вхідним вектором визначається за формулою 3.1.

$$d_j = \min_j \left\| \sum_{i=1}^{n} (x_i - w_{ij}) \right\|^2 = \min_j \|x - w_j\|^2 = \|x - w_c\|^2 \qquad (3.1)$$

- Визначення нейрона-переможця ініціює процес навчання, тобто модифікації області, яка включає сам нейрон та його деякий окіл, за ітераційною формулою 3.2. При цьому вектор, що описує нейрон-переможець, та вектори, що описують його сусідів, у сітці переміщуються в напрямку вхідного вектора. Це проілюстровано на рисунку 3.1.

$$w_j(n+1) = w_j(n) + \Phi_{cj}(n)\big[x(n) - w_j(n)\big], \qquad (3.2)$$

де $\Phi_{cj}(n) = \Phi\big(\|l_c - l_j\|, n\big)$ – функція сусідства, яка залежить від відстані між координатою l_c нейрона-переможця c та координатою l_j

поточного нейрона j. У якості функції відстані може бути застосована, наприклад, проста константа або Гаусова функція (3.3).

$$\Phi_{cj}(n) = \alpha(n) exp\left(-\frac{\|l_c - l_j\|^2}{2\sigma^2(n)}\right) \quad (3.3)$$

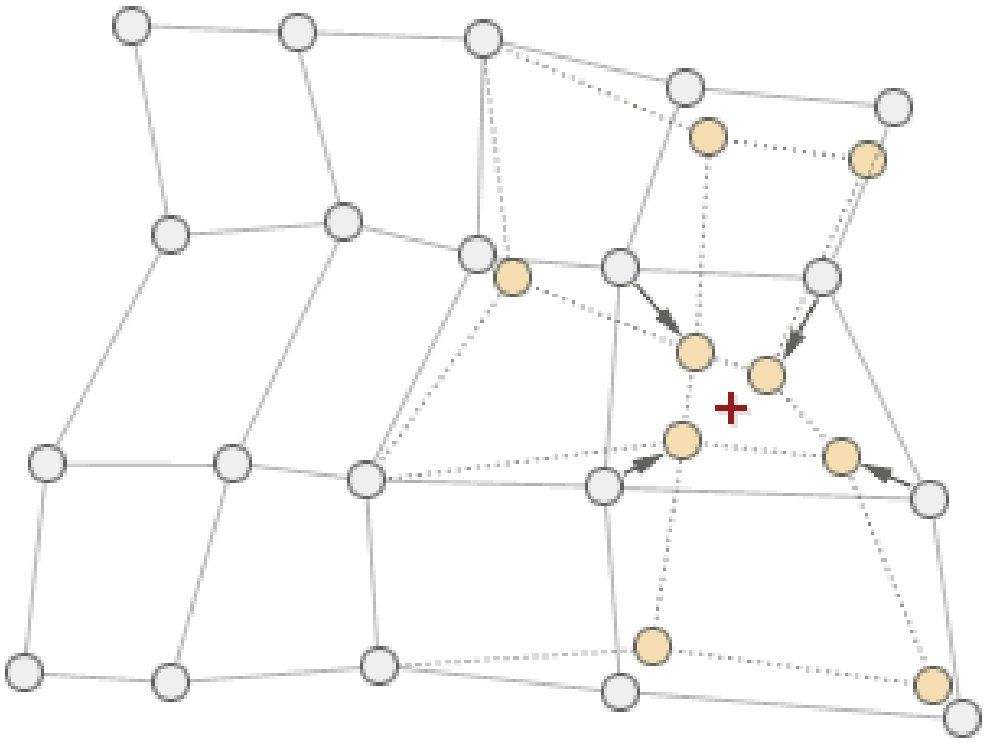

Рисунок 3.1 – Модифікація вагових коефіцієнтів нейрона-переможця та його сусідів

- Перевірка критерію збіжності за формулою 3.4.

$$\varepsilon_{t+1} = \frac{dist(D_{t+1}, D_t)}{D_t}, \quad (3.4)$$

де D_t – середнє значення відстаней між всіма векторами навчальної вибірки та всіма векторами мережі на кроці ітерації t. Якщо $\varepsilon_{t+1} \leq \mu$, то

навчання закінчене, інакше відбувається коригування коефіцієнтів та перехід на наступну ітерацію.

- Кластером буде група векторів, відстань між якими всередині групи менша, ніж відстань до сусідніх груп.

Перевагою нейронних мереж Кохонена перед іншими типами нейронних мереж є спрощення багатовимірної структури у двовимірну, некероване навчання, стійкість до зашумлених даних. Недоліки полягають в евристичності алгоритму, необхідності заздалегідь визначити кількість кластерів, розмірі отримуваних кластерів.

3.1.2 Центроїдні моделі. Кластеризація методом к-середніх

Кластеризація методом k-середніх (k-means) — один із найпопулярніших методів кластеризації, який полягає у впорядкуванні множини об'єктів у порівняно однорідні групи. Винайдений в 1950-х роках математиком Гуґо Штайнгаузом і майже одночасно Стюартом Ллойдом, даний метод отримав особливу популярність після виходу роботи МакКвіна.

Мета методу — розділити n спостережень на k кластерів так, щоб кожне спостереження належало до кластера з найближчим до нього середнім значенням. Метод базується на мінімізації суми квадратів відстаней між кожним спостереженням та центром його кластера. [16]

Алгоритм методу кластеризації за схемою к-середніх такий:

1) вибрати k точок як центри кластерів,

2) зіставити кожну точку набору з кластером, відстань до центру якого мінімальна;

3) переконатися, що в кожному кластері міститься хоча б одна точка, для цього кожний порожній кластер доповнити довільною точкою, що розташована «далеко» від центру власного кластера;

4) центр кожного кластера замінити середнім від елементів кластера;

5) повторити 2-4, поки не завершиться процес зміни центрів кластерів;

6) кінець.

Головні переваги методу k-середніх:

- простота та швидкість виконання;
- простота оптимізації алгоритму;
- помірні вимоги до пам’яті – O(KN) – та помірна вартість ітерації – O(KMN);
- метод k-середніх більш зручний для кластеризації великої кількості спостережень, ніж, наприклад, метод ієрархічного кластерного аналізу (у якому дендрограми стають перевантаженими і втрачають наочність).

Незважаючи на очевидні переваги методу, він має суттєві недоліки, а саме:

- порушення умови зв'язності елементів одного кластера, у зв’язку з чим розвиваються різні модифікації методу, а також його нечіткі аналоги (fuzzy k-means methods), у яких на першій стадії алгоритму допускається приналежність одного елемента множини до декількох кластерів (із різним ступенем приналежності);
- результат класифікації сильно залежить від випадкових початкових позицій кластерних центрів;
- алгоритм чутливий до викидів, які можуть викривлювати середнє;

- кількість кластерів фіксована і повинна бути заздалегідь визначена дослідником, а неправильна кількість кластерів може призвести до помилкових результатів;
- працює лише з Евклідовими величинами;
- час виконання алгоритму не обмежений, хоча із реальними проблемами він справляється успішно. [17]

Метод k-середніх є доволі простим і прозорим, тому успішно використовується у різноманітних сферах — маркетингових сегментаціях, геостатистиці, астрономії, сільському господарстві тощо. Незважаючи на неспроможність даного методу виявляти кількість кластерів та необхідність апріорних знань дослідника про це число, у зв'язку із простотою та швидкістю виконання даного алгоритму, він часто застосовується разом із методами валідації кластерного розбиття для декількох різних значень кількості кластерів, що розглянуті у наступному підрозділі.

3.1.3 Статистичні моделі. EM-алгоритм

В основі EM-алгоритму лежить припущення, що множина даних, яка досліджується, може бути змодельована за допомогою лінійної комбінації нормальних розподілів. Тобто припускають, що дані у кожному кластері розподілені за деяким нормальним законом. Це припущення може бути перефразоване таким чином, що дані належать до кожного кластеру, проте із різною ймовірністю. І метою даного методу є підбір таких розподілів для кожного кластеру. Очевидно, що точка буде віднесена до того кластеру, ймовірність з'явитись у якому у неї найвища.

Кожна ітерація алгоритму складається з двох кроків. На E-кроці (expectation) вираховується очікуване значення функції правдоподібності. На M-кроці (maximization) вираховується оцінка максимальної схожості, таким чином збільшується очікувана схожість, вирахувана на E-кроці. Потім це значення використовується для E-кроку на наступній ітерації. Алгоритм виконується до збіжності.

За певних обставин зручно розглядати EM-алгоритм як два кроки максимізації, що чергуються між собою. Розглянемо функцію 3.5

$$F(q,\theta) = E_q[\log L(\theta; x, Z)] + H(q) = \\ = -D_{KL}\left(q||p_{Z|X}(\cdot\,|x;\ \theta)\right) + \log L(\theta; x), \tag{3.5}$$

де q — розподіл ймовірностей неспостережуваних змінних Z,

$p_{Z|X}(\cdot\,|x;\ \theta)$ — умовний розподіл неспостережуваних змінних при фіксованих спостережуваних x і параметрах розподілення ймовірностей неспостережуваних змінних θ,

H — ентропія і

D_{KL} — відстань Кульбака — Лейблера.

Тоді кроки EM-алгоритму можна показати як наступні:

E(xpectation) крок:

Вибираємо q, щоб максимізувати F (3.6).

$$q^{(t)} = arg\max_q F\left(q, \theta^{(t)}\right) \tag{3.6}$$

M(aximization) крок:

Вибираємо θ, щоб максимізувати F (3.7).

$$\theta^{(t+1)} = arg\max_{\theta} F\left(q^{(\theta)}, \theta\right) \quad (3.7)$$

Алгоритм EM простий і легкий в реалізації, не чутливий до ізольованих об'єктів і швидко збігається при вдалій ініціалізації. Однак він вимагає для ініціалізації вказівки кількості кластерів k, що має на увазі наявність апріорних знань про дані. Крім того, при невдалій ініціалізації збіжність алгоритму може виявитися повільною або алгоритм може зупинитись у точці локального мінімуму і видавати квазіоптимальний результат.

3.1.4 Ієрархічні моделі

Ієрархічна кластеризація може бути агломераційною (висхідний підхід) або поділяючою (низхідний підхід). Відстань до кожного екземпляру розраховується з використанням деякої функції відмінності. Відстань між кластерами розраховується з використанням деякого критерію зв’язку. Кожен крок ієрархічної кластеризації створює новий набір кластерів з набору кластерів, отриманого на попередньому кроці.

Агломеративна ієрархічна кластеризація починається з набору кластерів, де кожен екземпляр належить своєму кластеру. На кожному кроці алгоритм з’єднує два найближчих кластери до тих пір, доки всі кластери не будуть об’єднані в один кластер, що містить всі екземпляри, тобто виконання даного алгоритму закінчується отриманням набору кластерів із одного кластера з усіма екземплярами. Поділяюча ієрархічна кластеризація працює у зворотному порядку – вона починається з набору кластерів з одним кластером, що містить всі екземпляри. На кожному етапі алгоритм рекурсивно розділяє кластери, використовуючи

деякий метод поділу, доки не досягне одного набору кластерів, що містить лише сінглтони, тобто де кожен екземпляр розташований в своєму власному кластері.

Результатом кластеризації є список, що містить набір кластерів і відповідні відмінності/відстані, що описують їх створення на кожному етапі алгоритму. Результат організований в ієрархічній формі, тобто, де кожен кластер посилається або на двох батьків, що були об’єднані для його створення (в агломераційному підході), або на два дочірніх елементи, отримані в результаті поділу кластеру (в розділювальному підході). Через їх ієрархічну природу результати кластеризації можуть бути представлені за допомогою дендрограми. [19]

3.1.4.1 AHC/AGNES

Агломеративна ієрархічна кластеризація (agglomerative hierarchical clustering) є популярним алгоритмом кластеризації, що послідовно об'єднує менші кластери в більші до отримання одного великого кластеру, що включає всі точки/об'єкти.

Постановка задачі для даного алгоритму включає в себе набір точок/об’єктів, метрику (формулу, що використовується для визначення відстані між двома точками), тип зв’язку (формулу, що використовується для визначення відстані між двома кластерами). В результаті виконання алгоритму отримується дендрограма, яку можна використовувати, щоб швидко отримати верхні K кластерів для будь-якого заданого K. [17]

Для заданого набору даних, що містить N точок, які необхідно кластеризувати, алгоритми агломеративної ієрархічної кластеризації

зазвичай починаються з N кластерів (кожна окрема точка даних є власним кластером). Алгоритм продовжується шляхом об'єднання двох окремих кластерів до більшого кластеру, доки не буде отриманий один кластер, що містить усі N точок даних. Очевидно, що алгоритм може мати інші критерії зупинки, такі як критерій завершення, коли отримується кластеризація, що містить визначену користувачем кількість кластерів (k).

Алгоритм AGNES (agglomerative nesting) кожної ітерації обирає два кластери для об'єднання, базуючись на найкоротшій Евклідовій відстані між кластерами, сформованими до сих пір.

Щоб знайти найближчу пару кластерів (тобто мінімальну відстань), AGNES шукає найкоротшу відстань в матриці відмінностей. Матриця відмінностей (DM) завжди оновлюється, тому що алгоритм кожної ітерації об’єднує найближчу пару кластерів у новий кластер, і такий кластер є новим записом матриці. Більш формально, на кожному рівні t, коли два кластери об'єднуються в один, розмір матриці відмінностей DM стає $(N - t) \times (N - t)$. DM_t випливає з DM_{t-1} шляхом:

а) видалення двох рядків та стовбців, що відповідають об'єднаним кластерам, і

б) додавання нового рядка і нового стовпчика, що містять значення відстаней між новоствореними кластерами і старими кластерами. [20]

Перевагами такого підходу до кластеризації є:

- можливість використовувати будь-яку метрику;
- можливість обирати між багатьма типами зв'язків;
- можливість працювати з верхніми K кластерами для будь-якого заданого K або з повною дендрограмою.

Недоліками є:

- вимоги до пам’яті $O(N^2)$ (алгоритм має зберігати декілька копій матриці відстаней $N \times N$), що робить алгоритм прийнятним лише для задач середнього масштабу (до 10 тисяч точок). Великомасштабні проблеми не можуть бути вирішені через обмежену пам’ять;
- час роботи $O(N^{2M})$ (де M – кількість функцій) [17].

Перевагою даного алгоритму є здатність послідовно проводити розбиття (об’єднання) на кластери і бути зупиненим щойно буде досягнена умова достатньої якості розбиття.

3.1.4.1.1 Типи метрик

Алгоритм агломеративної ієрархічної кластеризації може працювати з багатьма різними типами метрик. Серед інших, підтримуються такі метрики:

- класична Евклідова L_2;
- Чебишева L_{-inf};
- Манхеттен (міський квартал) L_0;
- кореляція Пірсона (включаючи абсолютну кореляцію);
- косинус метрика (в тому числі абсолютна косинус метрика);
- кореляція Спірмена (включаючи абсолютну кореляцію).

Метрика обирається при додаванні набору даних з точками кластеризації. [17]

3.1.4.1.2 Типи зв’язків

Метрика – це формула, що визначає відстань між двома точками. Але алгоритм також потребує задання деякої формули для визначення відстані між двома кластерами A і B, кожен з яких має декілька точок.

Формула, що визначає відстань між кластерами, як було сказано раніше, називається «типом зв’язку», а результати алгоритму сильно залежать від обраного типу зв’язку, який може бути, наприклад, одним із наступних:

- повний зв’язок – відстань між двома кластерами A та B є найбільшою відстанню між двома точками даних, з яких a належить A, а b належить B:

$$dist(A, B) = max(dist(a, b), a \in A, b \in B); \quad (3.8)$$

- одинарний зв’язок – відстань між двома кластерами A та B є найкоротшою відстанню між двома точками даних, з яких a належить A, а b належить B:

$$dist(A, B) = min(dist(a, b), a \in A, b \in B); \quad (3.9)$$

- незважений середній зв’язок – відстань між двома кластерами є середньою відстанню між точками даних кожного кластеру в одному кластері до кожної точки у другому кластері:

$$dist(A, B) = average(dist(a, b), a \in A, b \in B); \quad (3.10)$$

- середньозважений зв’язок;

- метод Уорда.

За замовчуванням використовується повний зв’язок, оскільки він дає найкращі результати (надійність, кількість кластерів). [17]

3.1.4.2 DIANA

Методи поділу кластерів також мають ієрархічну природу. Їх основна відмінність від агломеративних методів, описаних вище, полягає в тому, що вони діють в зворотному порядку. На кожному кроці метод поділу розділяє кластер на два менших, доки кожен кластер не міститиме лише один елемент. Це означає, що ієрархія знову будується за N-1 кроків, коли набір даних містить N об’єктів.

Більшість книг із кластерного аналізу приділяють мало уваги методам поділу, а більшість програмних пакетів взагалі не містять методів поділу. Основною причиною цього, очевидно, є обчислювальний аспект. На першому етапі агломераційного алгоритму розглядаються всі можливі об’єднання двох об’єктів, що призводить до

$$C_n^2 = \frac{n(n-1)}{2} \tag{3.11}$$

комбінацій. Це число росте з ростом n квадратично, тому воно стає великим, але обчислення все ще можливі. Алгоритм поділу, базований на тому ж принципі, буде починатись з розгляду усіх можливих поділів набору даних на дві непорожніх множини, що складає

$$2^{n-1} - 1 \tag{3.12}$$

можливостей. Останнє число росте експоненційно швидко і незабаром перевищує поточну оцінку кількості атомів у Всесвіті. Навіть для наборів даних середнього розміру такий підхід повного перечислення є обчислювально забороненим.

Тим не менш, можна побудувати методи поділу, які враховують не всі дочірні поділи, більшість із яких були б абсолютно недоречними в будь-якому випадку. Нижче описано алгоритм, базований на пропозиції Macnaughton-Smith (1964).

Всі набори даних, які можуть бути кластеризовані за допомогою агломераційного підходу, також можуть бути і проаналізовані з поділом. У більшості прикладів час обчислень просто збільшується вдвічі.

Дані складаються з матриці відстаней між об’єктами a, b, c, d, і e. Будучи поділяючим, алгоритм припускає, що об’єкти спочатку формують єдиний кластер { a, b, c, d, e}, як показано на рисунку 3.2.

	a	*b*	*c*	*d*	*e*
a	0.0	2.0	6.0	10.0	9.0
b	2.0	0.0	5.0	9.0	8.0
c	6.0	5.0	0.0	4.0	5.0
d	10.0	9.0	4.0	0.0	3.0
e	9.0	8.0	5.0	3.0	0.0

Рисунок 3.2 – Матриця відстаней між об’єктами a, b, c, d, і e

На першому кроці алгоритм має розбити набір даних на два кластери. Це робиться не за допомогою розгляду всіх можливих поділів, а з використанням деякої ітераційної процедури. Спочатку знаходять об’єкт, який є найвіддаленішим від всіх інших об’єктів. Щоб це зробити більш точно, необхідно ввести спосіб визначення відстані між об’єктом

та групою об’єктів. Як і в попередньому підході, для цих цілей може бути використана середня відстань, тому шукають об’єкт, для якого середня відстань до всіх інших об’єктів є найбільшою (коли таких об’єктів два, один із них обирають випадковим чином).

На цьому етапі створено групи {a} і {b, c, d, e}, але алгоритм не зупиняється. Для кожного об'єкту більшої групи обчислюють середню відстань до інших об'єктів і порівнюють із середньою відстанню до об'єктів «відколеної» групи.

У даному прикладі використано відстань до об'єкта b, який лежить далі від решти об'єктів з власної групи, ніж від відколених об'єктів. Тому об'єкт b змінює сторону, і звідси випливає нова відколена група – {a, b}, а залишкова група набуває вигляду {c, d, e}. На цьому етапі всі відстані стали від'ємними.

Наступного кроку розділяють найбільший кластер, тобто кластер з найбільшим діаметром (як і раніше, діаметр кластера – це найбільша відмінність між двома його об'єктами). Діаметр {a, b} дорівнює 2.0, а для {c, d, e) він дорівнює 5.0. Відповідно, описана вище процедура буде застосована до кластера {c, d, e}. [18]

Описаний вище алгоритм, якщо взяти до уваги предметну область, що розглядається у даній магістерській дисертації, може досягти достатньої якості розбиття за меншу кількість ітерацій, проте через нижчу швидкість виконання ітерацій він все одно програє агломеративній кластеризації.

3.1.5 Щільнісні моделі

3.1.5.1 DBSCAN

Алгоритм DBSCAN – один із найперших винайдених алгоритмів кластеризації щільнісним методом. В основі цього алгоритму лежать кілька визначень:

- ε-околицею об'єкта називається околиця радіусу ε деякого об'єкта;
- кореневим об'єктом називається об'єкт, ε-околиця якого містить не менше деякої мінімальної кількості MinPts об'єктів;
- об'єкт p безпосередньо щільно-досяжний з об'єкта q, якщо p знаходиться в ε-околиці q і q є кореневим об'єктом;
- об'єкт p щільно-досяжний з об'єкта q при заданих ε і MinPts, якщо існує послідовність об'єктів p_1, ..., p_n, де $p_1 = q$ і $p_n = p$, така що p_{i+1} безпосередньо щільно досяжний з p_i, $1 \leq i \leq n$;
- об'єкт p щільно-з'єднаний з об'єктом q при заданих ε і MinPts, якщо існує об'єкт o такий, що p і q щільно-досяжні з o.

Для пошуку кластерів алгоритм DBSCAN перевіряє ε-околицю кожного об'єкта. Якщо ε-околиця об'єкта p містить більше точок, ніж MinPts, то створюють новий кластер з кореневим об'єктом p. DBSCAN ітеративно збирає об'єкти безпосередньо щільно-досяжні з кореневих об'єктів, які можуть привести до об'єднання кількох щільно-досяжних кластерів. Процес завершується, коли ні до одного кластеру не може бути додано жодного нового об'єкта. Хоча, на відміну від більшості методів кластеризації, DBSCAN не вимагає заздалегідь вказувати число одержуваних кластерів, при застосуванні даного алгоритму виникає потреба у вказівках значень параметрів ε і MinPts, які безпосередньо впливають на результат кластеризації. Оптимальні значення цих

параметрів складно визначити, особливо для багатовимірних просторів даних.

3.1.5.2 HDBSCAN

HDBSCAN – це алгоритм кластеризації, розроблений Кампелло, Мулаві та Сандером у роботі [21]. Він розширяє DBSCAN, частково перетворюючи його на алгоритм ієрархічної кластеризації, і використовує техніку вилучення пласкої кластеризації, базовану на стабільності кластерів.

Він виконує DBSCAN для різних значень ε та інтегрує результат, щоб знайти кластеризацію, яка забезпечить найкращу стабільність в порівнянні з ε. Це дозволяє HDBSCAN знаходити кластери різної щільності (на відміну від DBSCAN) та бути більш стійким до вибору параметрів. [22]

На практиці це означає, що HDBSCAN одразу ж повертає гарну кластеризацію з мінімальним налаштуванням параметрів або без нього, проте все ще потребує визначення мінімального розміру кластерів.

Як описано в [23], можна розбити даний алгоритм на такі кроки:

- перетворення простору відповідно до щільності/розрідженості;
- побудова мінімального дерева протяжності зваженого по відстані графа;
- побудова ієрархії кластерів пов'язаних компонентів;
- скорочення ієрархії кластерів, базуючись на мінімальному розмірі кластера;
- вилучення стабільних кластерів з ущільненого дерева.

Щоб знайти кластери, необхідно знайти «острівці» більш високої щільності серед «моря» більш розрізнених шумів – і припущення про шум важливе: реальні дані можуть мати викиди, спотворення та шум. Ядром алгоритму кластеризації є кластеризація з одинарним зв'язком, і вона може бути досить чутливою до шуму. Даний алгоритм розроблений як стійкий до шуму, тому він забезпечує спосіб зменшення впливу шумів перед запуском одно-зв'язкового алгоритму.

На основі оцінки щільності, точки із більш низькою щільністю розглядаються як «море». Метою даного кроку є зробити ядро кластеризації більш стійким до шуму. Отже, враховуючи ідентифікацію «моря», «морські» точки мають бути більш віддаленими одна від одної та від «суші».

На практиці потрібна дуже недорога оцінка щільності, і найпростіше – це відстань до k-го найближчого сусіда. Оскільки є матриця відстаней для вхідних даних, то це той тип запиту, для якого гарно працюють kd-дерева. Дана відстань називається відстанню до ядра, визначеною для параметру k і точки x, і позначається як $\text{core}_k(x)$. Простий спосіб розбити точки з низькою щільністю (відповідно, більшою відстанню до ядра) – визначити нову метрику відстані між точками, яка називається відстанню взаємної досяжності. Відстань взаємної досяжності визначається як

$$d_{mreach-k}(a,b) = \max\{core_k(a), core_k(b), d(a,b)\} \qquad (3.13)$$

де $d(a,b)$ – це початкова метрика відстані між a і b.

При цьому метрично щільні точки (з низькою відстанню до ядра) лишаються на одній і тій самій відстані одна від одної, але більш розріджені точки відштовхуються так, щоб знаходитись як мінімум на відстані їх ядра від будь-якої іншої точки. Це ефективно «знижує рівень

моря», поширюючи розріджені «морські» точки, лишаючи «сушу» без змін. Застереження тут полягає в тому, що великі значення k інтерпретують більше точок, як ті, що знаходяться в «морі». У прикладі нижче використовується значення k, рівне п’яти. Для заданої точки можна намалювати коло відстані до ядра як коло, що торкається шостого найближчого сусіда (враховуючи саму точку), приблизно так, як показано на рисунку 3.3.

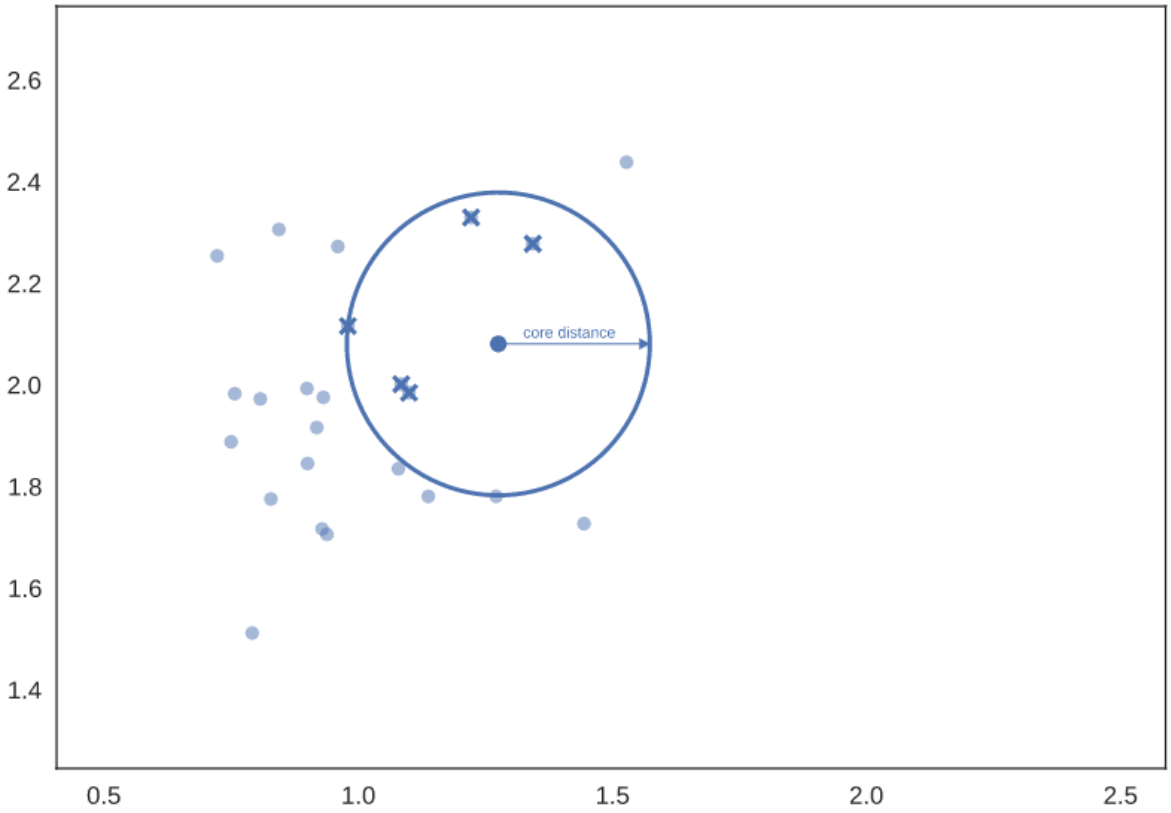

Рисунок 3.3 – Відстань до ядра синього кола

Обравши іншу точку, можна зробити те саме, але цього разу з іншим набором сусідів (один з них – це точка, яку було обрано попереднього разу). Даний крок проілюстровано на рисунку 3.4.

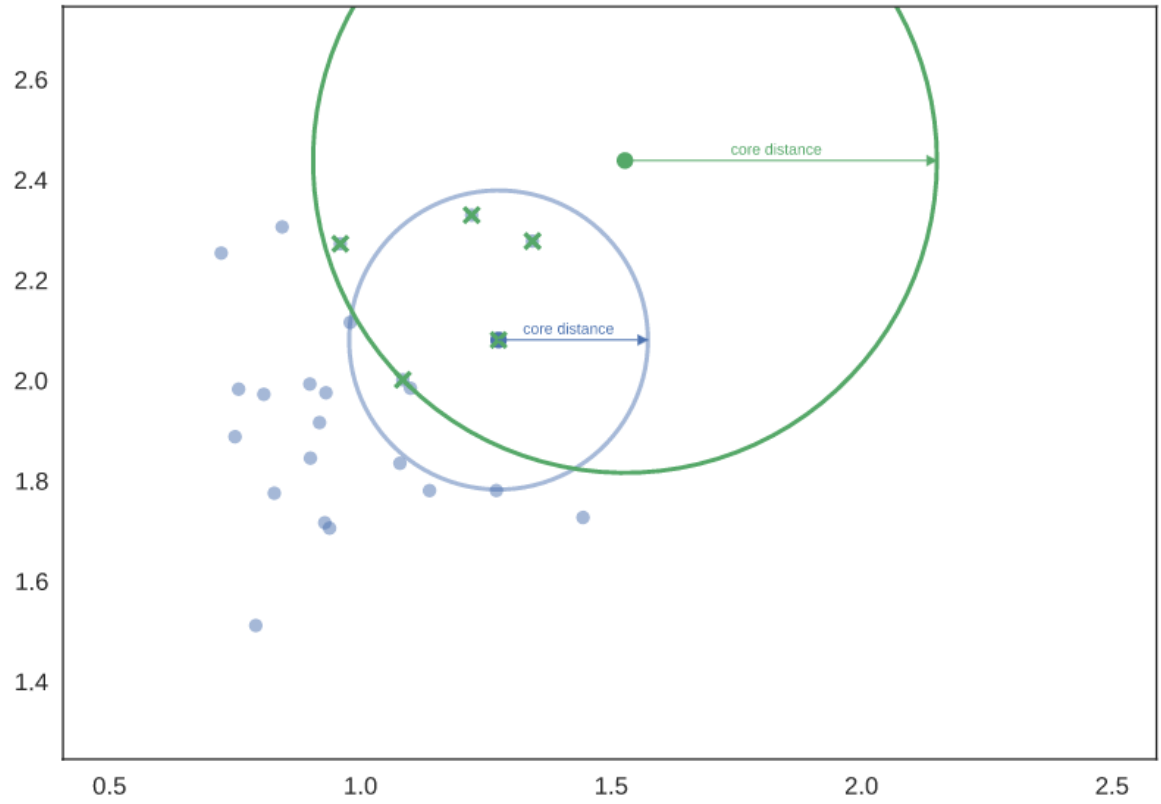

Рисунок 3.4 – Відстань до ядра зеленого кола

Далі цей крок можна повторити і втретє, для кращої точності вимірів, з іншою множиною шести найближчих сусідів, як показано на рисунку 3.5.

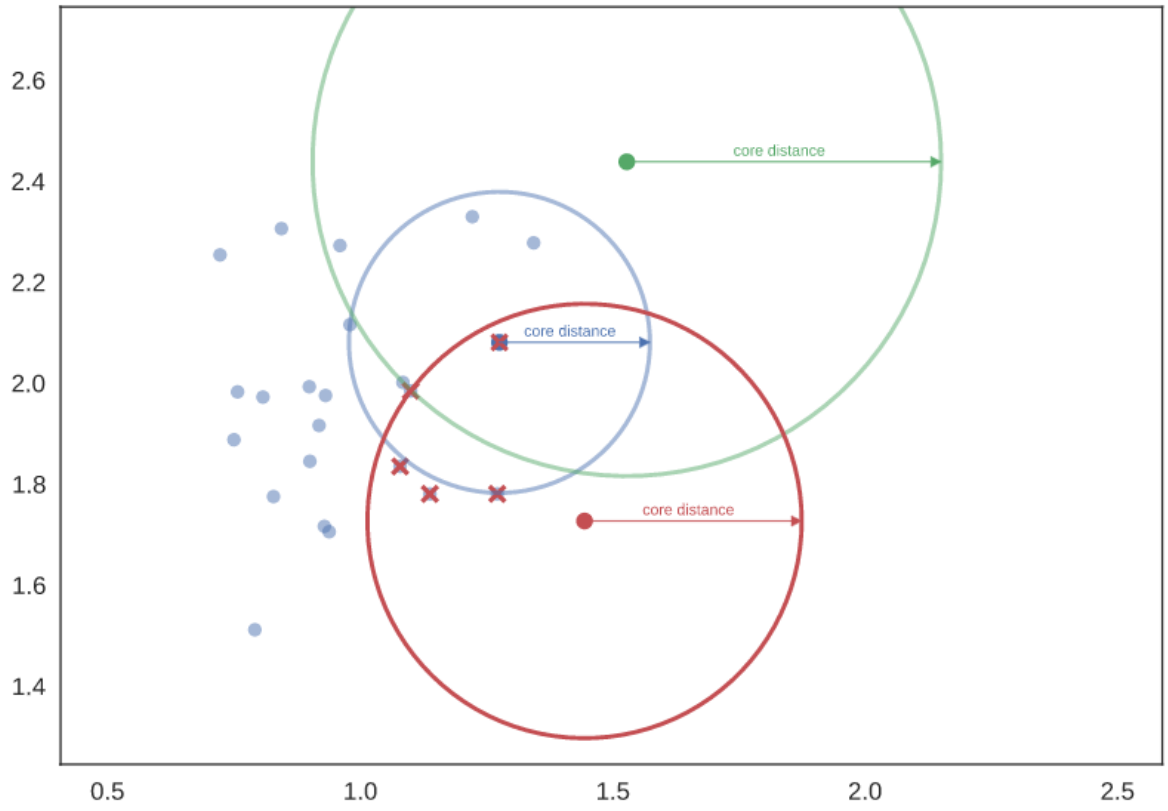

Рисунок 3.5 – Відстань до ядра червоного кола

Тепер для визначення відстані взаємної досяжності між синьою та зеленою точками, можна почати з позначення стрілкою відстані між зеленою та синьою точками (рисунок 3.6).

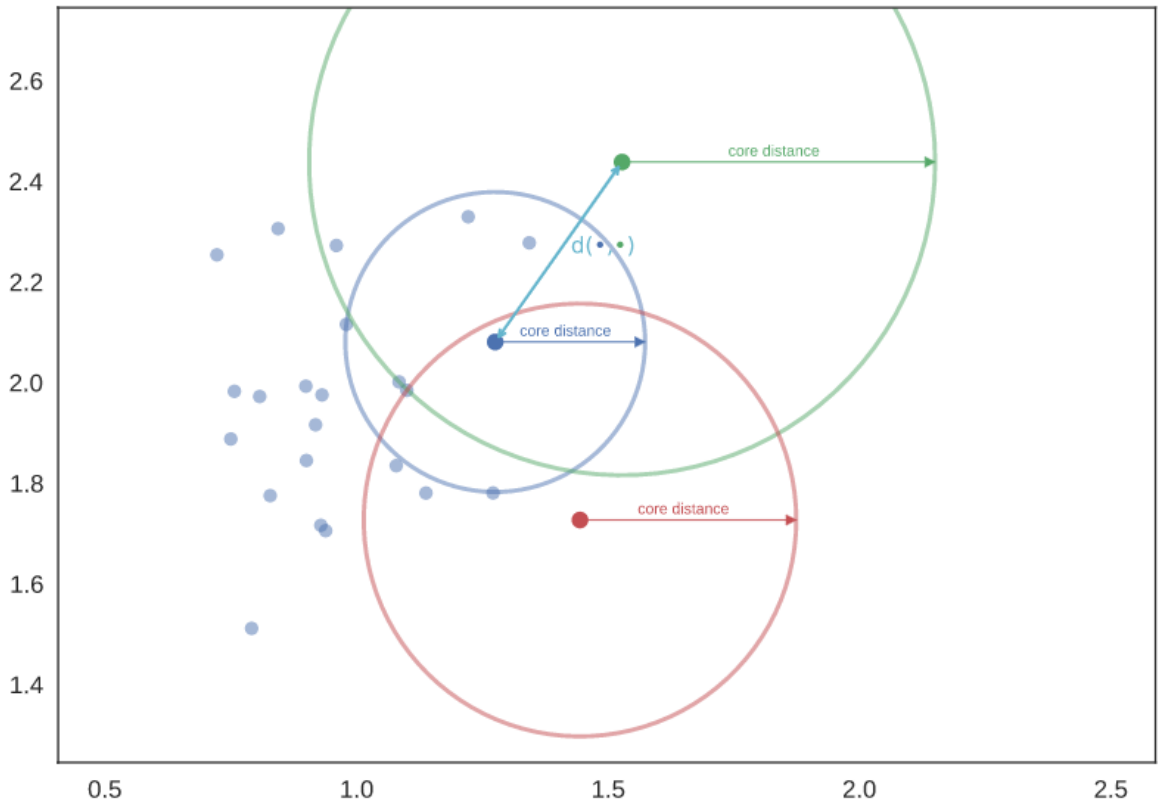

Рисунок 3.6 – Відстань між центрами синього та зеленого кіл

Вона перетинає синє коло, але не зелене коло – основна відстань для зеленого кола більша, ніж відстань між синьою та зеленою точками. Таким чином, потрібно відмітити відстань взаємної досяжності між синьою та зеленою точками як значення, що більше або дорівнює радіусу зеленого кола, що проілюстровано на рисунку 3.7.

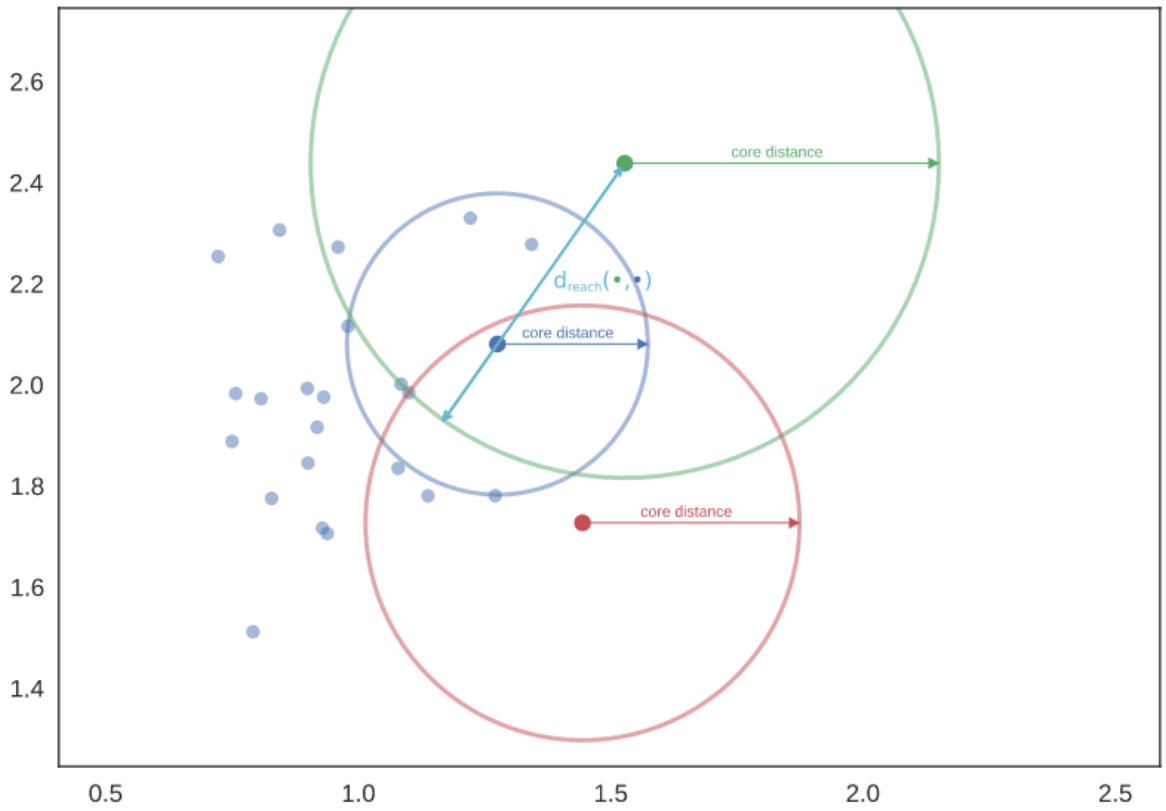

Рисунок 3.7 – Досяжність синього та зеленого кластерів дорівнює радіусу зеленого кола

З іншого боку, відстань взаємної досяжності від червоної до зеленої точки – це просто відстань між червоною та зеленою точками, оскільки ця відстань більша, ніж будь-яка відстань до ядра (тобто стрілка відстані проходить через обидва кола), як видно із рисунку 3.8.

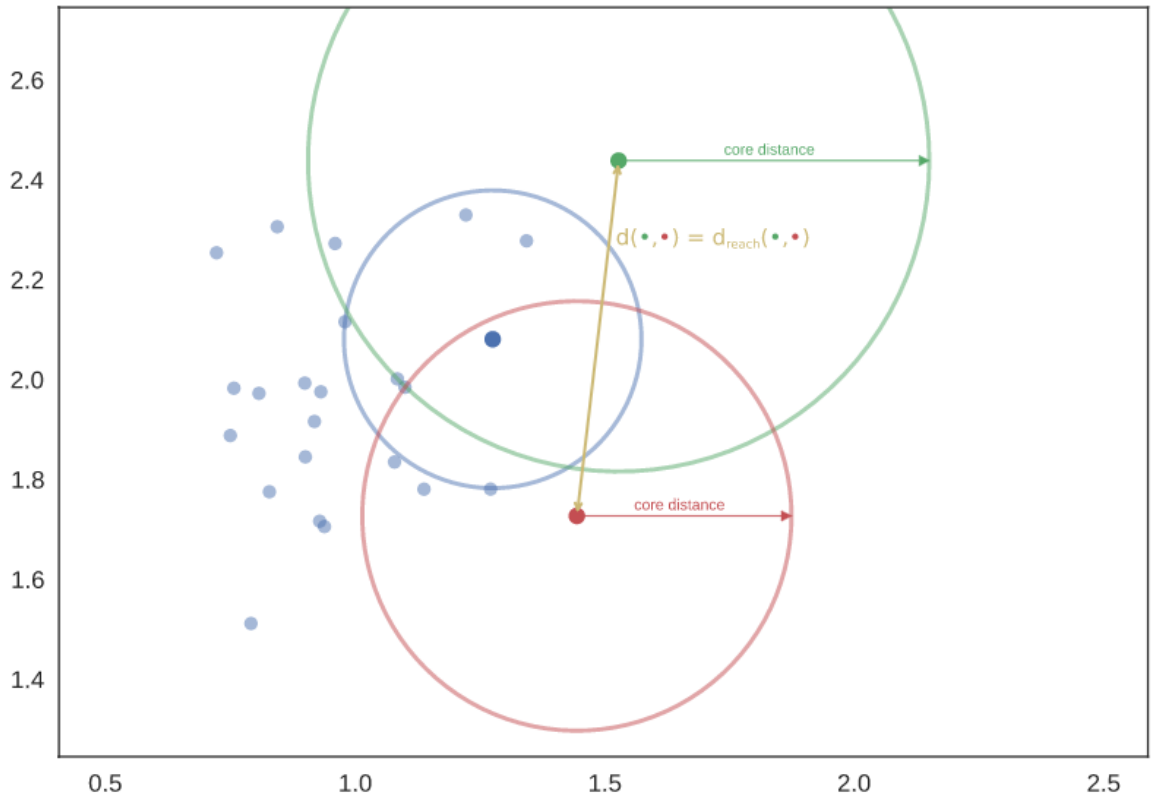

Рисунок 3.8 – Досяжність зеленого та червоного кола

В цілому, існує базова теорія, викладена у [24], що демонструє, що відстань взаємної досяжності як перетворення працює добре, дозволяючи кластеризації з одинарним зв'язком краще апроксимувати ієрархію рівнів множин будь-якого істинного розподілу щільності, з якого були взяті точки.

На основі даної метрики взаємної досяжності даних починають пошук «острівців» на щільних даних. Щільні області є відносними, і різні «острови» можуть мати різну щільність. Концептуально, дані розглядають як зважений граф з точками даних у вигляді вершин і ребрами між будь-якими двома точками, вага яких дорівнює відстані взаємної досяжності цих точок.

Далі розглядають граничні значення, починаючи з високого рівня і поступово знижуючи його. Відкидають будь-які ребра з вагою, вищою

за цю границю. У міру того, як опускаються ребра, граф починає роз'єднуватися на зв’язані компоненти. З часом створюється ієрархія пов’язаних компонентів (від повністю підключених до повністю відключених) на різних граничних рівнях.

На практиці, виконання даного алгоритму потребує дуже багато пам'яті: існує n^2 ребер, і не вигідно запускати алгоритм зв’язаних компонентів багато разів. Оптимальніше знайти мінімальний набір ребер, такий щоб будь-яке ребро з набору викликало роз’єднання компонентів. Але при цьому потрібно, щоб не було краю з меншою вагою, який міг би з’єднати компоненти. На щастя, теорія графів забезпечує саме таку можливість: мінімальне дерево протяжності графа.

Побудувати мінімальне дерево протяжності можна дуже ефективно за допомогою алгоритму Прима – дерево будується по одному ребру за раз, завжди додаючи ребро з найменшою вагою, яке поєднує поточне дерево з вершиною, що ще не міститься в дереві.

У випадку, коли дані знаходяться в метричному просторі, можна використовувати ще швидші методи для побудови мінімального дерева протяжності.

Враховуючи мінімальне дерево протяжності, наступний крок – перетворити його в ієрархію пов’язаних компонентів. Це легше за все зробити у зворотному порядку: відсортувати ребра дерева по відстані (у порядку зростання), а потім виконати ітерації, створюючи новий об’єднаний кластер для кожного ребра. Єдина складна частина полягає в тому, щоб ідентифікувати два кластери, які будуть об’єднані кожним ребром, але це досить просто зробити з допомогою структури даних об’єднання-пошуку. [25] Результат можна спостерігати у вигляді дендрограми, як видно на рисунку 3.9 нижче.

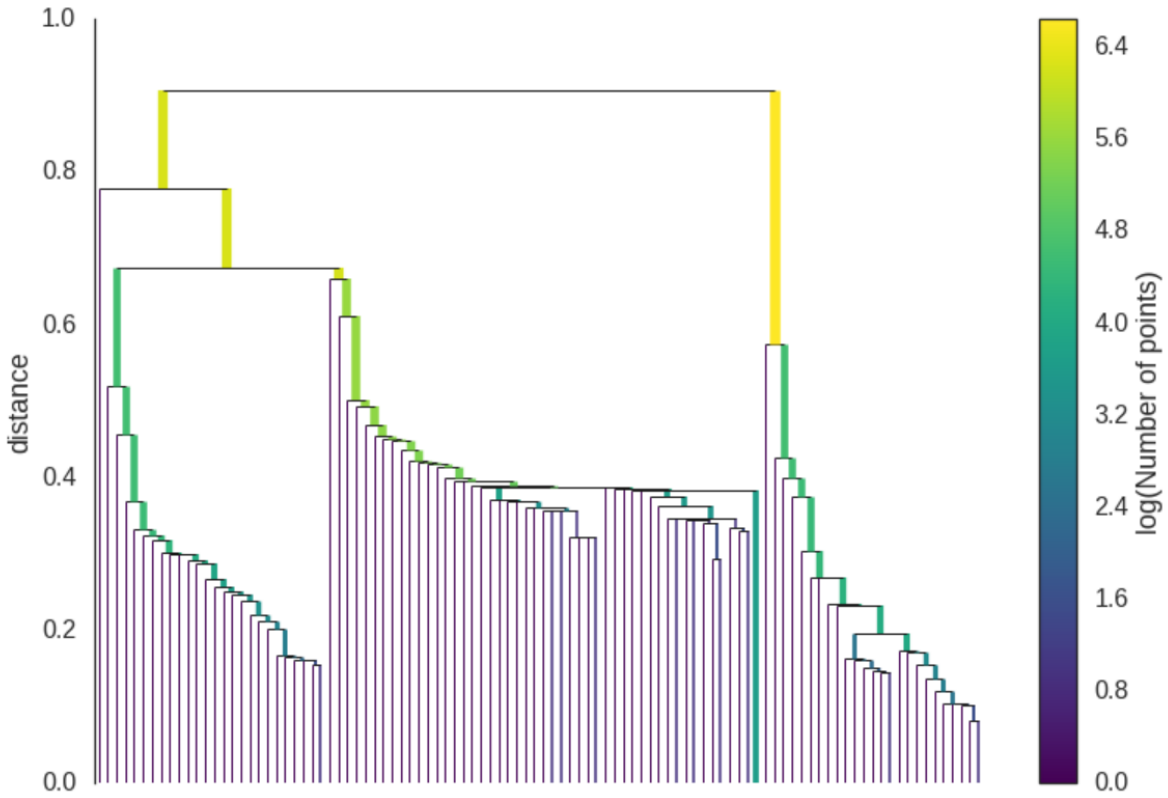

Рисунок 3.9 – Дендрограма як результат кластеризації

На даному моменті алгоритм із одинарним зв’язком закінчується. Далі необхідно отримати плаский набір кластерів замість ієрархічного представлення. Перший крок до визначення кластерів – спрощення отриманої ієрархії до меншого дерева. Для цього визначають мінімальний розмір кластера, який передають як параметр до алгоритму. На кожному кроці перегляду ієрархії аналізують поділ поточного кластеру на два. Якщо кластер, що відділяється, має менше точок, ніж мінімальний розмір кластера, ці точки позначають як такі, що випадають з кластера і дані про них зберігаються у батьківському кластері. Якщо ж кластер, що відділяється, містить більше точок, ніж мінімальний розмір кластера, то такий поділ вважають дійсним і відображають у новому дереві. [25] Отриманий результат проілюстровано на рисунку 3.10.

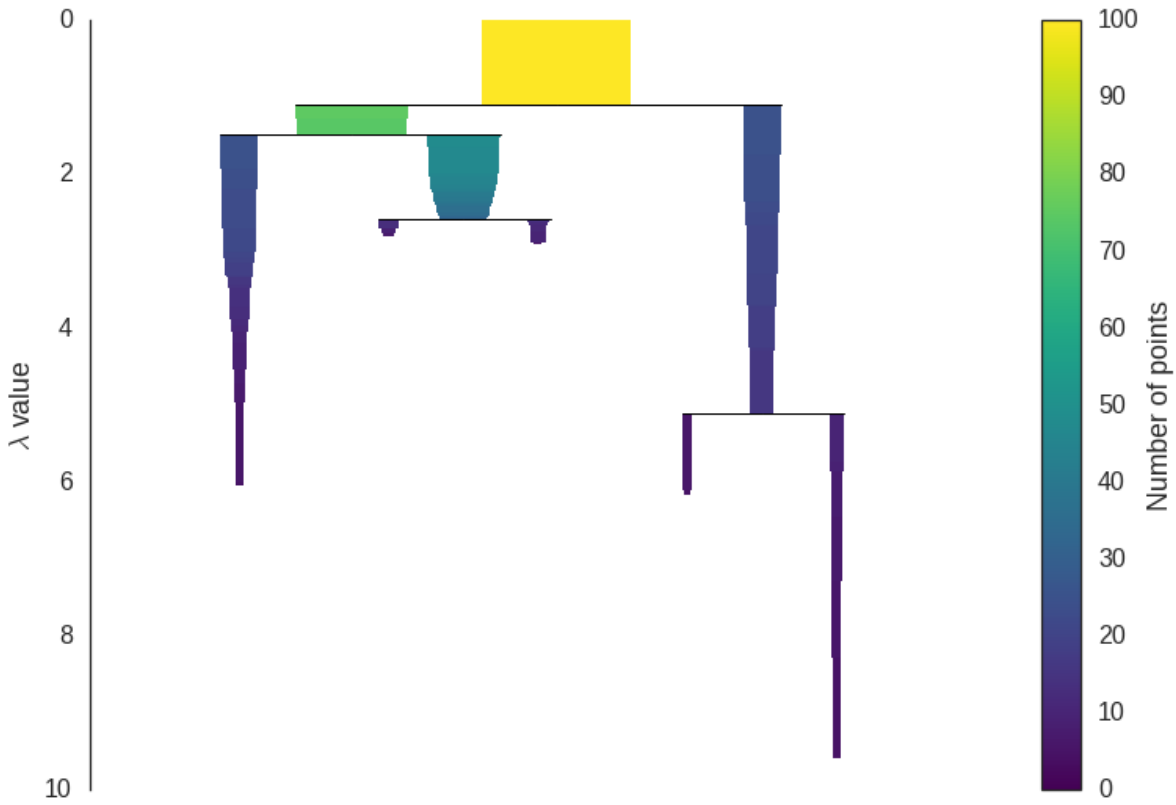

Рисунок 3.10 – Дендрограма дерева кластеризації

Інтуїтивно хочеться залишити ті кластери, які мали найдовший термін життя протягом процесу кластеризації, адже решта, скоріше за все, є просто артефактами застосованого алгоритму. Таким чином необхідно обрати ті області, які займають найбільшу площу на рисунку 3.10.

Даний метод є досить складним, містить багато змінних та логічних частин, має висок обчислювальну складність та складність реалізації.

3.1.6 Графові моделі. HCS

Алгоритм кластеризації HCS (Highly Connected Subgraphs, сильнозв’язні підграфи), також відомий як Highly Connected Clusters/Components/Kernels, – це алгоритм, базований на зв'язності графів. Даний метод спочатку представляє дані про подібність об’єктів у графі подібності, а після цього знаходить всі сильно зв'язані підграфи як кластери. Алгоритм не потребує жодних попередніх припущень про кількість кластерів. Цей алгоритм був опублікований Ерезом Хартувом та Роном Шаміром в 1998 році.

Метою даного методу є групування елементів у множини, що не перетинаються, або кластери, на основі подібності між елементами таким чином, щоб елементи одного кластеру були дуже схожі один на одного (однорідність), в той час як елементи з різних кластерів мали низьку схожість один з іншим (поділ). Графік подібності є одною з моделей для представлення схожості між елементами і, у свою чергу, полегшує створення кластерів. Щоб побудувати граф схожості, необхідно представити елементи у виді вершин та додавати ребра між вершинами, коли значення подібності між ними перевищує деяке граничне значення.

У графі подібності справедливо, що чим більше ребер існує для даного числа вершин, тим більше такий набір вершин схожий між собою. Іншими словами, якщо спробувати роз’єднати граф подібності, видаливши ребра, то чим більше ребер потрібно видалити, доки граф стане роз’єднаним, тим більше схожих вершин є в цьому графі. Мінімальний розріз – це мінімальний набір ребер, без яких граф стане роз’єднаним.

Алгоритм кластеризації HCS знаходить усі підграфи з n вершинами, оскільки мінімальний розріз цих підграфів містить більше, ніж n/2 ребер, та ідентифікує їх як кластери. Такий підграф називається сильнозв’язним підграфом (HCS). Поодинокі вершини не вважаються кластерами і групуються в множину сінглтонів S.

З огляду на граф подібності G(V, E), алгоритм кластеризації HCS перевіряє, чи є він сильнозв'язним, і якщо так, то повертає G, інакше використовує мінімальний розріз G, щоб розбити G на два підграфи H і H' та рекурсивно виконати алгоритм кластеризації HCS на H і H'.

Час роботи алгоритму HCS обмежено значенням (3.14)

$$N \times f(n, m), \tag{3.14}$$

де $f(n, m)$ – це часова складність обчислення мінімального розрізу в графі з n вершинами та m ребрами,

N – кількість знайдених кластерів, $N \ll n$.

Кластери, отримані за допомогою алгоритму кластеризації HCS, мають властивості однорідності та розподілу.

Діаметр кожного високозв’язного графа дорівнює максимум двом. Число ребер у сильнозв’язному підграфі квадратичне. Число ребер, що видаляються кожної ітерації алгоритму HCS, максимально лінійне. Це чітко вказує на однорідність, оскільки єдина найкраща можливість з точки зору діаметру полягає в тому, що кожні дві вершини кластера поєднані ребром, що є занадто суворим, а також задачею складності NP. Це також вказує і на розподіл, оскільки число ребер, що видаляються кожної ітерації алгоритму HCS, максимально лінійне за відстанню основного підграфу, на відміну від квадратичного числа ребер в кінцевих кластерах.

Елементи, залишені в якості сінглтонів у процесі початкової кластеризації, можуть бути «прийняті» кластерами на основі подібності з кластером. Якщо максимальна кількість сусідів до конкретного кластеру досить велика, то його можна додати в цей кластер.

Коли вхідний граф має вершини з низькими ступенями, запуск алгоритму не вартий того, адже він вимагає великих обчислювальних витрат і не є інформативним. [26]

3.2 Методи валідації розбиття на кластери

Термін кластерна валідація, або перевірка якості розбиття даних на кластери, описує процедуру оцінки якості результатів алгоритму кластеризації. На жаль, такі алгоритми, як, наприклад, к-середніх та розбиття навколо медоїд, вимагають апріорних знань про кількість кластерів перед запуском, проте дане число може бути суб’єктивним і залежати одночасно від характеру самих даних, а також вибору функцій відстані, подібності та інших параметрів у методах, що використовуються. Виходячи із цього, існують спеціальні методи для визначення оптимальної кількості кластерів, які описані далі.

3.2.1 Метод «ліктя»

Найвідоміший метод оцінки якості розбиття називається «методом ліктя» і полягає у розрахунку загальної внутрішньокластерної варіації (суми квадратів відстаней до центру кластера) як функції від кількості

кластерів. Отримані значення зображуються на графіку. Найкращою вважається кількість кластерів, для якої отримана крива має виражений згин, схожий на згин ліктя чи коліна.

Даний метод дуже простий і наочно показує, як збільшення кількості кластерів перестає впливати на дані. Проте даний метод може давати неоднозначні результати та помилки. Як альтернатива для нього у роботі [18] Кауфманом та Россеєвим був запропонований метод середнього силуету.

3.2.2 Метод середнього силуету

Силуетний аналіз визначає середню відстань між кластерами. Графік силуету показує, як близько кожна точка з одного кластеру розташована відносно точок із інших кластерів.

Для кожної точки і ширина силуету s_i обчислюється за алгоритмом:

- для кожної точки і обчислюється середня відмінність a_i між і та всіма іншими точками кластеру, до якого вона належить;
- для всіх інших кластерів C, до яких точка і не належить, обчислюється відмінність d(i, C) для кожного елемента C; найменша з таких d(i, C) визначається за формулою 3.15

$$b_i = \min_C d(i, C), \tag{3.15}$$

де b_i – відмінність між точкою і та її сусіднім кластером, тобто найближчим кластером, до якого вона не належить;

- врешті силуетна ширина і визначається за формулою 3.16.

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \tag{3.16}$$

Точки, для яких $s_i \to 1$, кластеризовані дуже добре, точки, для яких дана метрика приблизно дорівнює нулю, лежать між двома кластерами, точки із від'ємним s_i віднесені до невідповідного кластеру [27].

Найкращою кількістю кластерів вважається та, для якої середній силует усіх точок максимальний.

3.2.3 Індекс Данна

Даний метод, описаний у роботі [28], вимірює відношення найкоротшої відстані між об'єктами із різних кластерів до найдовшої внутрішньокластерної відстані. Індекс Данна може набувати значень від нуля до нескінченності і більший результат вказує на кращу кластеризацію. Його метою є знаходження кластерів, які є компактними та щільними і при цьому достатньо віддаленими один від одного. Алгоритм даного методу такий:

- для кожного кластеру визначити відстань між кожним об'єктом даного кластеру та об'єктами з інших кластерів;
- вибрати найменшу із цих попарних відстаней як міжкластерну розподіленість;
- для кожного кластеру визначити відстані між об'єктами даного кластеру та обрати найбільшу як внутрішньокластерну зв'язність, або максимальний діаметр кластерів;

- визначити індекс Данна (D) за формулою 3.17.

$$D = \frac{min.separation}{max.diameter} \tag{3.17}$$

3.2.4 Within-between коефіцієнт

Розрахунок даного коефіцієнту базується на розрахунку суми квадратів відстаней всередині кластерів та між кластерами, звідки і походить його назва. Обґрунтування алгоритму наведено у роботі [29].

Коефіцієнт розраховується за формулою 3.18.

$$\rho = \frac{sum\ of\ squares\ within\ clusters}{sum\ of\ squares\ between\ clusters} \cdot \frac{number\ of\ clusters}{number\ of\ points} \tag{3.18}$$

Існує безліч (більше 30) менш популярних метрик для визначення найкращої кількості кластерів, які також базуються на внутрішньокластерній зв’язності, або компактності, та міжкластерній розподіленості. Більшість із них схожі і дещо варіюються у визначеннях даних величин. Наприклад, компактність може визначатися як найбільша відстань між двома точками всередині одного кластеру, найбільша відстань від точки до центру її кластеру і так далі. Також і сама відстань може визначатися за різними метриками. Усі ці методи можуть легко бути модифікованими і комбінованими між собою, залежно від обраного методу кластеризації та характеру даних, із якими доводиться працювати.

3.3 Порівняльний аналіз методів кластеризації даних

Важливими для даної магістерської дисертації критеріями якості алгоритмів кластеризації визначено обчислювальну складність алгоритмів та, відповідно, їх швидкодію, необхідність апріорних знань про дані, залежність від ініціалізації алгоритму, здатність коректно працювати з викидами або сінглтонами. Такі характеристики, як форма кластерів та інтуїтивна можливість візуалізації результату кластеризації, не є актуальними для даного дослідження у зв'язку із вибором предметної області.

Результати проведеного за даними критеріями порівняння наведено у таблиці 3.1.

Таблиця 3.1 – Порівняння алгоритмів кластеризації за визначеними критеріями

Назва алгоритму	Обчислювальна складність	Необхідність апріорних знань	Залежність від ініціалізації	Здатність коректно працювати з викидами
Самоорганізаційні мапи Кохонена	$O(N^2K)$	Ні	Так	Так
К-середні	O(NK)	Так (k)	Так	Ні
EM	$O(NKq^2)$	Так (k)	Так	Так
Агломеративний	$O(N^3)$	Ні	Ні	Так
DBSCAN	$O(N^2)$	Так (ε, MinPts)	Ні	Ігнорує
HCS	O(KNM)	Ні	Ні	Ігнорує

Алгоритм самоорганізаційних мап, за дослідженням Т. Кохонена [15], потребує не менше 500 запусків для досягнення статистичної вірогідності. Аналогічний недолік має також ЕМ-алгоритм.

Агломеративний алгоритм чудово справляється з викидами, за виключенням випадку використання одинарного зв'язку.

Таким чином, на основі даних таблиці 3.1, можна скласти таблицю оцінок розглянутих алгоритмів за визначеними вище критеріями (таблиця 3.2).

Таблиця 3.2 – Оцінка алгоритмів

Назва алгоритму	Обчислювальна складність	Необхідність апріорних знань	Залежність від ініціалізації	Здатність коректно працювати з викидами	Сума оцінок
Самоорганізаційні мапи Кохонена	0	1	0	1	2
К-середні	1	0	1	0	2
ЕМ	0	0	0	1	1
Агломеративний	0.5	1	1	1	3.5
DBSCAN	1	0	1	0	1
HCS	0.5	1	1	0	2.5

З отриманої таблиці можна зробити висновок, що найбільш застосовним для розв'язку поставлених у даній магістерській дисертації задач, на основі визначених критеріїв, є алгоритм агломеративної ієрархічної кластеризації. Другу за розміром оцінку отримав алгоритм HCS, від якої не дуже відрізняються оцінки нейронних мереж Кохонена

та алгоритму к-середніх. EM-алгоритм та DBSCAN виявилися зовсім не застосовними до розв’язку задачі кластеризації помилок тестування.

3.4 Модифікація алгоритму кластеризації даних

Для подальшої реалізації системи кластеризації помилок тестування на основі порівняльного аналізу за обраними критеріями було обрано агломеративний алгоритм. Даний алгоритм легко модифікується шляхом вибору способу обчислення таких величин:

- відстань між двома точками;
- відстань між двома кластерами;
- відстань від точки до кластера, якому вона не належить;
- координати центру кластера;
- компактність кластера;
- критерій вибору оптимальної кількості кластерів та закінчення роботи алгоритму.

Висновок до розділу 3

У даному розділі розглянуто та проаналізовано такі наявні методи кластеризації даних, як нейронні моделі, центроїдні моделі (включаючи кластеризацію методом к-середніх), статистичні моделі (включаючи EM-алгоритм), ієрархічні моделі (включаючи агломеративний та поділяючий підходи), щільнісні моделі на прикладі DBSCAN та HDBSCAN, графовий алгоритм HCS. Також у даному розділі розглянуто основні методи вибору найкращої кількості кластерів. У підрозділі 3.3 визначено критерії оцінки розглянутих алгоритмів кластеризації даних, виходячи із предметної області магістерської дисертації, та проведено порівняльний аналіз вищевказаних алгоритмів. Найкращим для розв’язку поставленої задачі є агломеративний алгоритм, до якого будуть застосовані модифікації визначення відстаней.

РОЗДІЛ 4 АРХІТЕКТУРА СИСТЕМИ КЛАСТЕРИЗАЦІЇ

У даному розділі обґрунтовано вибір мови програмування, застосованої у розробці інтелектуальної системи кластеризації помилок тестування, проаналізовано архітектуру даної системи, описано принцип її роботи.

4.1 Обґрунтування вибору мови програмування

У зв’язку з тим, що найпоширенішою операційною системою серед користувачів усього світу, а відповідно, і найпоширенішою системою, що використовується при тестуванні програмного забезпечення, є Microsoft Windows, розроблений у результаті даного дослідження програмний продукт орієнтований саме на неї.

Відповідно, платформою для розробки програмного продукту обрана платформа .NET від компанії Microsoft, з якою сумісні усі машини, що працюють під ОС Windows. Вона має такі переваги:

- об'єктно-орієнтованість;
- кешування;
- простота підтримки;
- простота розробки;
- конзистентність;
- моніторинг;
- широкий функціонал.

Для імплементації інтелектуальної системи кластеризації помилок тестування обрано мову програмування C# та засоби .NET Framework.

C# – це сучасна об’єктно-орієнтована мова програмування, розроблена компанією Microsoft в межах її платформи .NET за ініціативою Андерса Гейлзберга. Основними властивостями даної мови є:

- об’єктно-орієнтованість;
- простота;
- швидкодія;
- безпечні типи – код має лише права виконання, а не зміни;
- сумісність – можливість взаємодіяти із кодом, написаним на інших мовах;
- розширюваність та простота внесення змін;
- структурованість – можливість розділяти розв’язок великої проблеми на маленькі модулі.

Основними перевагами мови C# є:

- об’єктно-орієнтованість;
- крос-платформеність – працює всюди, де є .NET фреймворк;
- автоматичний збір сміття – автоматичне звільнення пам’яті, що більше не може бути використана програмою;
- простота розробки;
- підтримка;
- велика спільнота розробників.

4.2 Аналіз вимог користувача

У таблиці 4.1 наведені історії користувача (user story). На основі даних вимог до програмного продукту розроблена його архітектура, описана у наступному підрозділі.

Таблиця 4.1 – Історії користувача

As a user	I want to input trx file	So that I can get clustering
As a user	I want to receive list of clusters including test names and messages	So that I can see clustering result
As a system	I want to serialize input data	So that I have data objects
As a system	I want to vectorize text messages	So that I have digital vectors for clustering
As a system	I want to get clusterings	So that I can find patterns in data
As a system	I want to evaluate clusterings	So that I can find the best clustering
As a user	I want to have logging	So that I can see the progress of program execution

4.3 Аналіз архітектури програмного продукту

Програмний продукт складається із 5 модулів: Deserialization, Vectorization, Clustering.Algorithm, Clustering.Evaluation, Logging. На рисунку 4.1 зображена діаграма зв’язків між модулями.

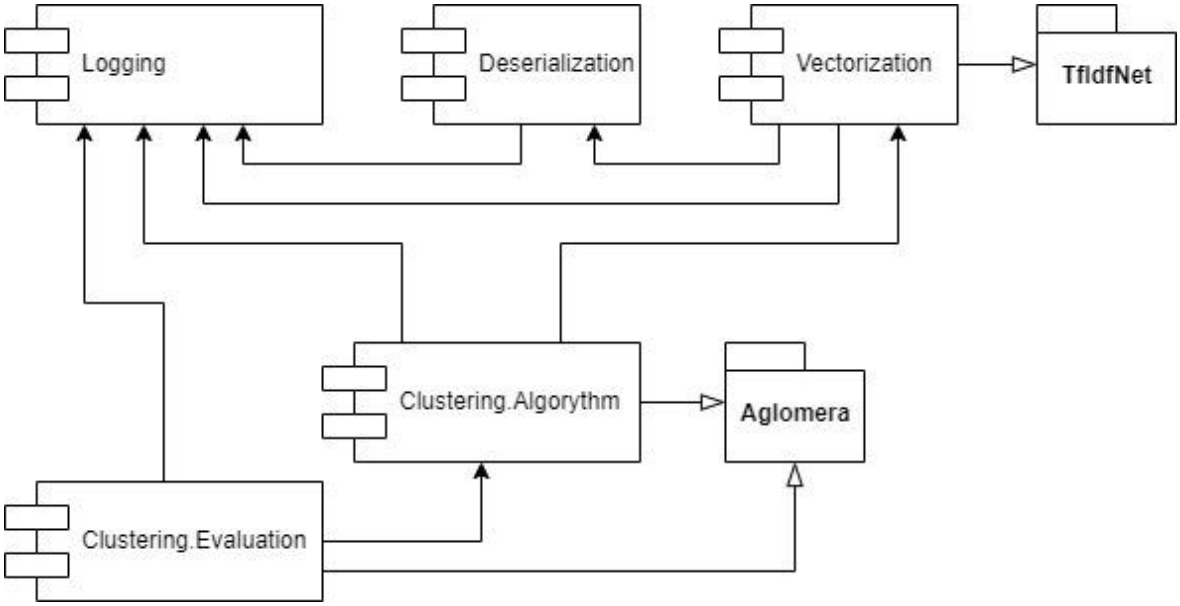

Рисунок 4.1 – Діаграма зв’язків між модулями

На рисунку 4.2 зображена діаграма класів модулю десеріалізації. Даний модуль трансформує вхідні дані у форматі xml в колекцію об’єктів та обирає лише необхідні для подальшого аналізу дані. Також даний модуль можна легко розширити для підтримки вхідних даних у форматі json.

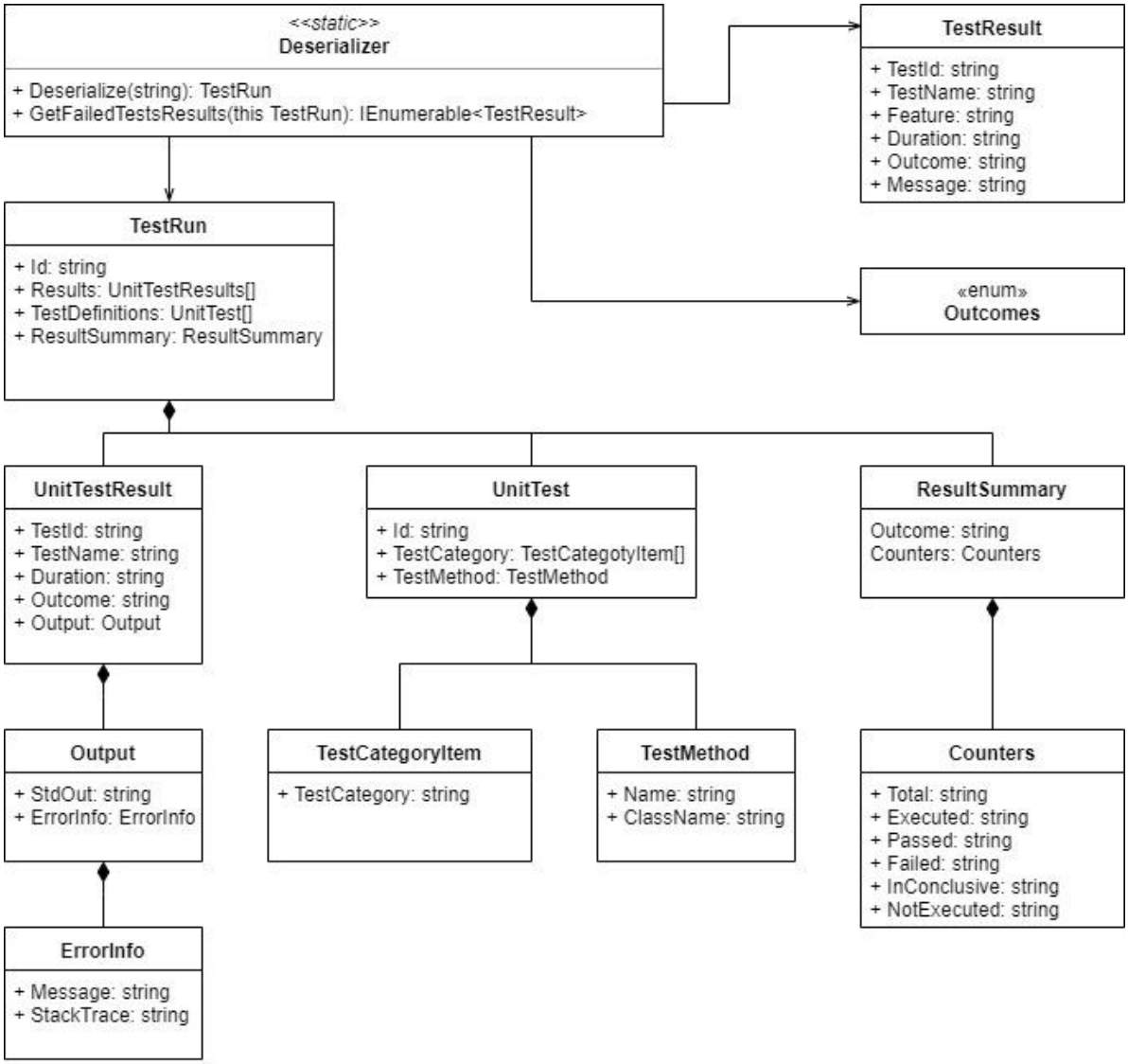

Рисунок 4.2 – Діаграма класів модулю десеріалізації

На рисунку 4.3 зображена діаграма класів модулю векторизації. Даний модуль реалізує функціонал, що трансформує дані, отримані попереднім модулем, у числові вектори. Модуль реалізує модифікований алгоритм TF-IDF, містить специфічний словник стоп-слів.

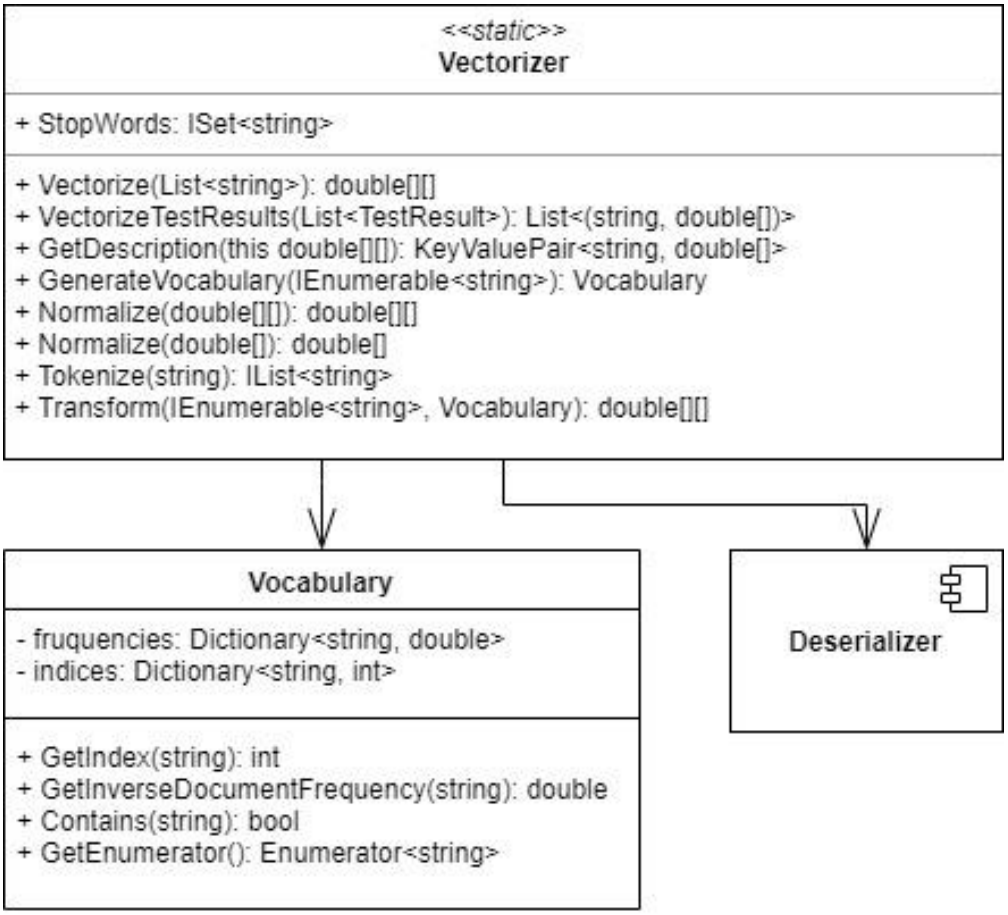

Рисунок 4.3 – Діаграма класів модулю векторизації

На рисунку 4.4 зображена діаграма класів модулю алгоритму кластеризації. Даний модуль реалізує агломеративний підхід. Реалізовано два різних підходи до визначення відстаней між точками (DataPoint – евклідова відстань, ManhattanDataPoint – відстань міських кварталів), реалізовано одинарний, повний, середній та центроїдний критерії зв’язності.

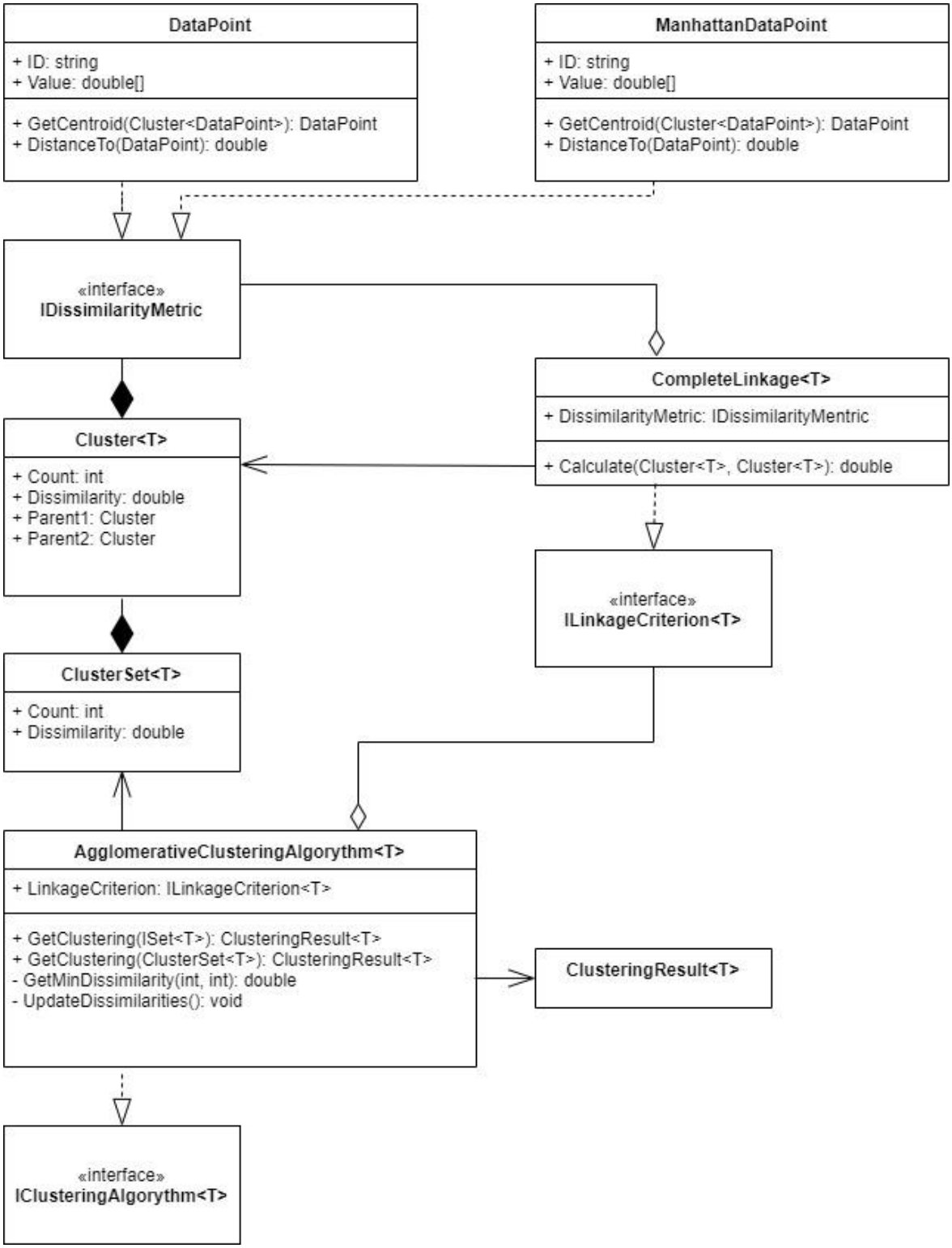

Рисунок 4.4 – Діаграма класів модулю кластеризації

На рисунку 4.5 зображена діаграма класів модулю оцінки якості кластеризації. Даний модуль, на основі даних, отриманих в результаті

кластеризації, проводить оцінку кожного з розбиттів за критеріями середнього силуету та Данна.

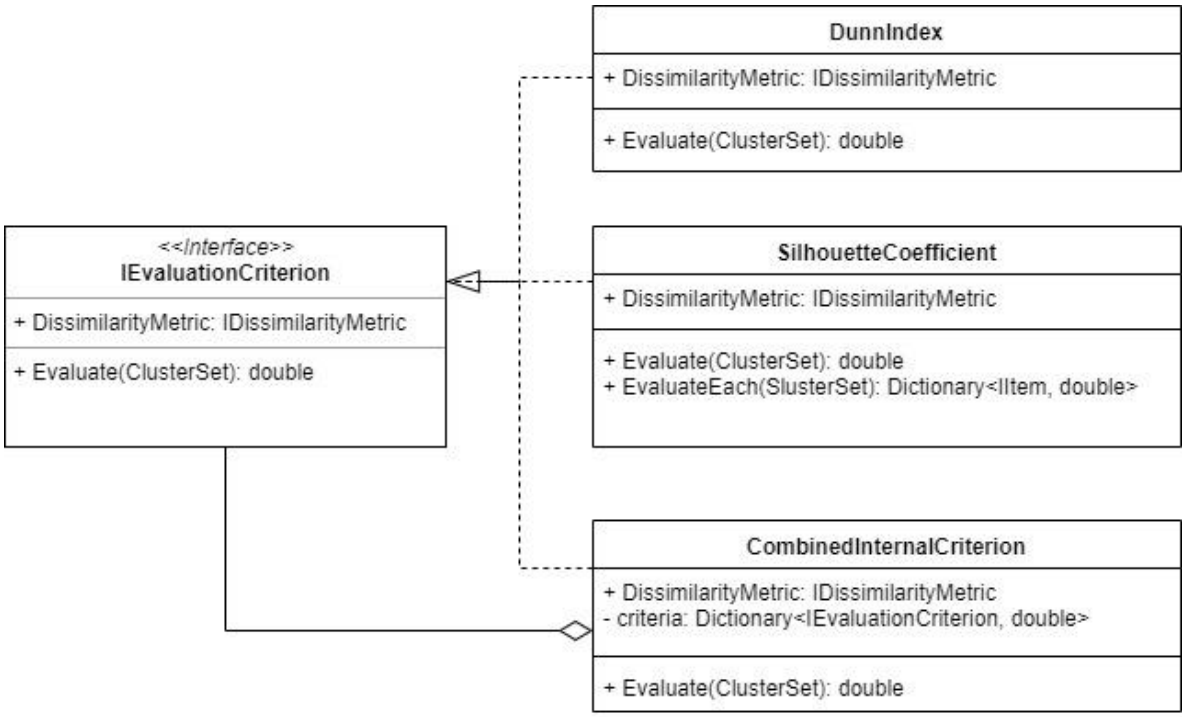

Рисунок 4.5 – Діаграма класів модулю оцінки якості кластеризації

Логування реалізоване з виведенням в консоль.

Для запуску програмного продукту необхідно викликати відповідні методи кожного з модулів у порядку десеріалізація – векторизація – кластеризація – оцінка на вхідних даних.

Наступним кроком розробки програмного забезпечення запропоновано використати вищеописані модулі у REST сервісі з інтуїтивно зрозумілим API для запуску кластеризації даних.

Висновок до розділу 4

У даному розділі обґрунтовано вибір засобів для розробки програмного продукту, розглянуто вимоги користувача до функціоналу розроблюваної системи та проаналізовано її архітектуру з наведенням діаграм класів.

Програму реалізовано засобами мови C#. Пакет програми складається із 5 модулів, які виконують атомарні функції, наприклад, десеріалізація, векторизація, кластеризація. Запуск системи відбувається шляхом виклику відповідних методів розроблених бібліотек.

РОЗДІЛ 5 АНАЛІЗ РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО ПРОДУКТУ

У даному розділі проведено порівняння та евристичний аналіз результатів роботи системи кластеризації помилок залежно від методів обчислення відстаней, зв’язків та якості розбиття.

Для аналізу було обрано 3 набори результатів тестування, що містять 80, 150 та 300 тестів зі статусом Failed (неуспішний результат). Для кожного із даних наборів було запущено алгоритм кластеризації з такими модифікаціями:

а) метрика, або відстань між об’єктами:

1) Евклідова:

$$d(X,Y) = \sqrt{\sum_i (x_i - y_i)^2}, \tag{5.1}$$

де i – кількість координат у векторах X та Y;

2) мангеттенська:

$$d(X,Y) = \sum_i |x_i - y_i| \tag{5.2}$$

б) зв’язок, або відстань між кластерами:

1) середній;

2) повний;

3) одинарний;

в) критерій якості розбиття:

1) індекс Данна;

2) коефіцієнт силуету;

3) within-between коефіцієнт, базований на сумі квадратів.

У таблиці 5.1 наведені результати виконання програми та евристична оцінка якості отриманих кластерів.

Таблиця 5.1 – Порівняння результатів виконання програми

1	2	3	4	5	6	7
80 результатів						
Metric	Linkage		criterion	clusters	distance	heuristic
Euclidean	Average	Elapsed time	0.0000877			
		Dunn	3,027	3	0,98	good
		Silhouette	0,893	3	0,98	good
		WithinBetween	0,071	3	0,98	good
	Complete	Elapsed time	0.0001041			
		Dunn	3,027	3	0,98	good
		Silhouette	0,893	3	0,98	good
		WithinBetween	0,071	3	0,98	good
	Single	Elapsed time	0.0000825			
		Dunn	3,027	3	0,98	good
		Silhouette	0,893	3	0,98	good
		WithinBetween	0,071	3	0,98	good
Manhattan	Average	Elapsed time	0.0000945			
		Dunn	10,753	3	0,98	good
		Silhouette	0,970	3	0,98	good
		WithinBetween	0,003	3	0,98	good
	Complete	Elapsed time	0.0000948			
		Dunn	10,753	3	0,98	good
		Silhouette	0,970	3	0,98	good
		WithinBetween	0,003	3	0,98	good
	Single	Elapsed time	0.0000952			
		Dunn	10,753	3	0,98	good
		Silhouette	0,970	3	0,98	good
		WithinBetween	0,003	3	0,98	good
150 результатів						
Euclidean	Average	Elapsed time	0.0024729			
		Dunn	1,474	2	7,21	bad
		Silhouette	0,794	7	4,24	good
		WithinBetween	0,410	7	4,24	good
	Complete	Elapsed time	0.0023517			
		Dunn	1,474	2	7,21	bad

Кінець таблиці 5.1

1	2	3	4	5	6	7
		Silhouette	0,794	7	4,24	good
		WithinBetween	0,410	7	4,24	good
	Single	Elapsed time	0.0229557			
		Dunn	1,474	2	7,21	bad
		Silhouette	0,794	7	4,24	good
		WithinBetween	0,410	7	4,24	good
Manhattan	Average	Elapsed time	0.0016234			
		Dunn	1,412	7	4,24	good
		Silhouette	0,825	7	4,24	good
		WithinBetween	0,090	7	4,24	good
	Complete	Elapsed time	0.0015627			
		Dunn	1,412	7	4,24	good
		Silhouette	0,825	7	4,24	good
		WithinBetween	0,090	7	4,24	good
	Single	Elapsed time	0.0027119			
		Dunn	1,412	7	4,24	good
		Silhouette	0,825	7	4,24	good
		WithinBetween	0,090	7	4,24	good
300 результатів						
Euclidean	Average	Elapsed time	0.0070010			
		Dunn	1,426	22	3,78	moderate
		Silhouette	0,461	2	4,73	bad
		WithinBetween	0,144	24	14,02	bad
	Complete	Elapsed time	0.0065401			
		Dunn	1,426	22	3,78	moderate
		Silhouette	0,461	2	4,73	bad
		WithinBetween	0,144	24	14,02	bad
	Single	Elapsed time	0.0072197			
		Dunn	1,426	22	3,78	moderate
		Silhouette	0,461	2	4,73	bad
		WithinBetween	0,144	24	14,02	bad
Manhattan	Average	Elapsed time	0.0076192			
		Dunn	2,186	22	9,74	moderate
		Silhouette	0,576	14	34,02	good
		WithinBetween	0,003	24	4,22	bad
	Complete	Elapsed time	0.0081675			
		Dunn	2,186	22	9,74	moderate
		Silhouette	0,525	15	34,02	moderate
		WithinBetween	0,003	24	4,22	bad
	Single	Elapsed time	0.0088435			
		Dunn	2,186	22	9,47	good
		Silhouette	0,508	15	29,07	moderate
		WithinBetween	0,003	24	4,22	bad

На основі даної таблиці можна зробити такі висновки:

- Обрані критерії якості розбиття дають різні оцінки залежно від характеру вхідних даних. Таким чином не виявлено найкращого критерію і не рекомендовано покладатися на значення лише одного критерію.
- Обрані критерії оцінки якості розбиття добре працюють із невеликими наборами даних, проте не дають адекватного результату, коли кількість вхідних даних перевищує 300. Евристично досліджено, що обрані на основі даних оцінок розбиття містять надто багато схожих кластерів. Запропоновано обрати інший, не досліджений в рамках даної роботи, критерій оцінки якості розбиття на кластери.
- Для невеликого обсягу вхідних даних (до 300 помилок) можна використовувати комбінований критерій, що базується на трьох розглянутих.
- Отримані в результаті виконання алгоритму кластери та час його роботи практично не залежать від вибору типу зв’язності.
- Час виконання алгоритму збільшується при збільшенні кількості вхідних даних. В середньому, повний зв'язок дає найкращий результат з точки зору часу.
- Отримані кластери залежать від обраної метрики. Визначено, що для характеру обраної предметної області краще підходить не загальноприйнята Евклідова метрика, а відстань міських кварталів.

Висновок до розділу 5

У даному розділі розглянуто та проаналізовано результати роботи розробленої системи з різними наборами вхідних даних (в якості вхідних даних було обрано справжні результати тестування із кількістю помилок 80, 150 та 300) та із різними модифікаціями параметрів, на яких базується алгоритм.

Розглянуто такі види метрик, як евклідова та мангеттенська, та визначено, що для даних такого характеру, як і ті, що використовувались у даному експерименті, краще підходить друга, тобто мангеттенська, або відстань міських кварталів.

Розглянуто такі типи зв’язків, як одинарний, повний та середній, і визначено, що суттєвої різниці у використанні різних типі немає, хоча повний зв’язок дає дещо кращі результати з огляду на час.

Розглянуто такі критерії оцінки якості розбиття на кластери, як коефіцієнт середнього силуету, індекс Данна та метрика, що базується на сумі квадратів внутрішньо- та міжкластерних відстаней. Евристично визначено, що дані критерії забезпечують вибір найкращого результату лише при невеликій (до 300) кількості повідомлень. Запропоновано розглянути інші способи оцінки якості кластеризації, що не розглянуті у даній магістерській дисертації.

РОЗДІЛ 6 СТАРТАП-ПРОЕКТ

У даному розділі описаний маркетинговий аналіз стартап-проекту, який включає в себе систему кластеризації помилок тестування, задля визначення принципової можливості його ринкового впровадження та можливих напрямків цього впровадження.

6.1 Опис ідеї технології

Таблиця 6.1 ілюструє зміст ідеї та можливі базові потенційні ринки, в межах яких потрібно шукати групи потенційних клієнтів.

Таблиця 6.1 – Опис ідеї стартап-проекту

Зміст ідеї	Напрямки застосування	Вигоди для користувача
Система для кластеризації помилок групує отриманий список помилок з пробігу тестів і класифікує кластери за вже відомими помилками	Тестувальниками як частина аналізу результатів регресійного тестування	Пришвидшує аналіз, так як дозволяє деякі кластери відкидати, у деяких кластерах – аналізувати лише один тест
	Тестувальниками як частина системи регулярного репортингу результатів тестування	Зменшує кількість ручної роботи зі складання звітів

У наступній таблиці (6.2) подається порівняння техніко-економічних властивостей ідеї відносно існуючих аналогів. Єдиним доступним програмним продуктом, який надає інтелектуальний аналіз помилок, є ReportPortal. Доцільним є порівняння саме модулів, які аналізують помилки.

Таблиця 6.2 – Визначення сильних, слабких та нейтральних характеристик ідеї проекту

1	2	3	4	5	6	7
№ п/п	Техніко-економічні характеристики ідеї	Товари/концепції конкурентів		W (слабка сторона)	N (нейтральна сторона)	S (сильна сторона)
		Мій проект	Конкурент			
1.	Економіч-ні	Безкошто-вно, сумнівна перспектива підтримки	Безкоштовно, велика команда підтримки	Підтримка вимагає значних затрат з моєї сторони	Ціна однакова	Швидкість підтримки

Кінець таблиці 6.2

1	2	3	4	5	6	7
2.	Техні-чні	Кластеризація (виявлення кластерів в некласифіко-ваних помилках)	Класифікація	Швидкість	Якість	Ширша функція
3.	Наді-йності	Постійна стабільна робота	Багато скарг на те, що інтелектуаль-ний аналіз не працює	Лише 1 людина для технічної підтримки		
4.	Адап-тив-ність	Модуль можна використову-вати з будь-якою графічною оболонкою	Використовує ться лише як складова ReportPortal	Не має графічної оболонки		Порта-бельний

Визначення характеристик та властивостей товару є підґрунтям для формування його конкурентоспроможності.

6.2 Технологічний аудит ідеї проекту

В межах даного підрозділу проводиться аудит технологій, за допомогою яких можна реалізувати ідею проекту.

У таблиці 6.3 наводиться опис необхідних для розроблення проекту технологій.

Таблиця 6.3 – Технологічна здійсненність ідеї проекту

1	2	3	4	5
№ п/п	Ідея проекту	Технології її реалізації	Наявність технологій	Доступність технологій
1.	Мова програмування	C#, .NET Framework	Наявна	Доступна, Community Edition
2.	Кластеризація помилок	Бібліотека для кластеризації	Жодна не працює достатньо якісно, необхідно розробити	
3.	Класифікація помилок	Бібліотека для класифікації	Наявна	Доступна або частково доступна, Nuget-пакет або python-пакет

За результатами аналізу таблиці можна зробити висновок, що технологічна реалізація проекту можлива та доцільна з використанням названих вище технологій.

6.3 Аналіз ринкових можливостей запуску стартап-проекту

У даному підрозділі визначено ринкові можливості, які можна використати під час ринкового впровадження проекту, та ринкових загроз, які можуть перешкодити реалізації проекту, проведено планування напрямів розвитку проекту із урахуванням стану ринкового середовища, потреб потенційних користувачів та пропозицій проектів-конкурентів.

У таблиці 6.4 наведено дані щодо наявності попиту, обсягу та динаміки розвитку ринку.

Таблиця 6.4 – Попередня характеристика потенційного ринку стартап-проекту

№ п/п	Показники стану ринку (найменування)	Характеристика
1.	Кількість головних гравців, од.	1
2.	Динаміка ринку (якісна оцінка)	Зростає
3.	Наявність обмежень для входу (вказати характер обмежень)	Немає
4.	Специфічні вимоги до стандартизації та сертифікації	Немає

Таким чином, проаналізувавши таблицю 6.4, можна зробити висновок, що даний ринок, за попереднім оцінюванням, є в міру привабливим для входження нового проекту.

У наступній таблиці, 6.5, визначаються потенційні групи клієнтів, їх характеристики, а також сформований орієнтовний перелік вимог до програмного продукту.

Таблиця 6.5 – Характеристика потенційних клієнтів стартап-проекту

1	2	3	4	5
№ п/п	Потреба, що формує ринок	Цільова аудиторія (цільові сегменти ринку)	Відмінності у поведінці різних потенційних груп клієнтів	Вимоги споживачів до товару
1.	Прискорення аналізу результатів виконання тестів	Тестува-льники-автома-тизатори	Залежно від кількості тестів, розміру логів та кількості різноманітних помилок	Стабільність роботи, надійність результатів класифікації, швидкість аналізу
2.	Зменшення кількості помилок для подальшого аналізу тестувальника ми	Тестува-льники-автома-тизатори		Групування всіх однакових помилок в групи, групи повинні бути повними, але при цьому не містити зайвих тестів

Таким чином, найімовірнішими потенційними клієнтами є тестувальники-автоматизатори, чиїми основними вимогами є швидкість і стабільність роботи системи.

Нижче проведено аналіз ринкового середовища. У таблиці 6.7 описані фактори, що сприяють ринковому впровадженню проекту, а в таблиці 6.6 наведено опис факторів, що йому перешкоджають. Фактори в таблицях наведені в порядку зменшення значущості.

Таблиця 6.6 – Фактори загроз

№ п/п	Фактор	Зміст загрози	Можлива реакція компанії
1.	Конкуренція	Поява нового конкурента на ринку з аналогічним продуктом	Підвищення ефективності роботи продукту
2.	Попит	Не вдається розробити унікальний метод, який би можна було застосовувати для будь-який вхідних даних	Розробка максимально універсального продукту

Таблиця 6.7 – Фактори можливостей

№ п/п	Фактор	Зміст можливості	Можлива реакція компанії
1.	Open-source	Допомога інших розробників з покращення програмного продукту	Прийняти допомогу
2.	Сторонні бібліотеки	Поява бібліотек, які можуть збільшити ефективність роботи програмного продукту	Проаналізувати час, затрачений на оновлення системи і профіт, отриманий від цього, і в результаті аналізу прийняти рішення
3.	Технічні	Поява нових технологій та алгоритмів у сфері кластерного аналізу, які будуть більш ефективними для реалізації даної системи	Активне використання наявних рішень; захист інтелектуальної власності

Нижче проведено аналіз пропозиції, у таблиці 6.8 визначено загальні риси конкуренції ринку.

Таблиця 6.8 – Ступеневий аналіз конкуренції на ринку

Особливості конкурентного середовища	В чому проявляється дана характеристика	Вплив на діяльність підприємства (можливі дії компанії, щоб бути конкурентоспроможною)
Тип конкуренції – чиста	Велика кількість методів, частина з яких є запатентованою інтелектуальною власністю	Звертати увагу на якість та універсальність методів
За рівнем конкурентної боротьби – глобальний	Проект не прив’язується до жодних географічних чинників	Акцент на розповсюдження серед працівників великих корпорацій
За галузевою ознакою – внутрішньогалузева	Конкуренцію складають подібні системи для аналізу помилок	Акцентувати увагу на якість роботи
За характером конкурентних переваг – нецінова	Акцент робиться на якості послуги	Підвищення якості роботи алгоритму та універсальності системи

У наступній таблиці (6.9) проведено більш детальний аналіз конкуренції в галузі за моделлю 5 сил М. Портера.

Таблиця 6.9 – Аналіз конкуренції в галузі за М. Портером

Складові аналізу	Прямі конкуренти в галузі	Потенційні конкуренти	Клієнти	Товари-замінники
	ReportPortal	Open-source проекти тестувальників-автоматизаторів для вирішення тієї ж проблеми	Тестувальники-автоматизатори	Мануальна робота з аналізу
Висновки	Інтенсивність конкуренстної боротьби з боку прямих конкурентів висока, проте наявність аналогу для системи, яка працює не дуже добре, привабить багато користувачів	Можливості входу в ринок є, конкуренція помірна	Клієнти повністю диктують умови роботи, потребами в швидкості і кількості оброблюваної інформації	Мануальна робота завжди можлива, проте дуже повільна і виснажлива для тестувальни-ків

За результатами аналізу таблиці можна зробити висновок щодо принципової можливості роботи на ринку з огляду на конкурентну ситуацію: конкуренція помірна й очікувана, виходити на ринок можна.

Розроблений програмний продукт повинен бути стабільним, швидким, швидко аналізувати велику кількість логів помилок тестування за раз.

На основі аналізу конкуренції, проведеного у таблиці 6.9, а також з урахуванням характеристик ідеї проекту із таблиці 6.2 та факторів маркетингового середовища із таблиць 6.6 та 6.7 можна визначити перелік факторів конкурентоспроможності. У таблиці 6.10 наведено їхній аналіз.

Таблиця 6.10 – Обґрунтування факторів конкурентоспроможності

№ п/п	Фактор конкурентоспроможності	Обґрунтування (наведення чинників, що роблять фактор для порівняння конкурентних проектів значущим)
1.	Фактор часу	Ідея є частково новою, для прийняття ідеї та втілення її у життя потенційними конкурентами знадобиться той же час
2.	Фактор новизни товару	Початковий успіх продукту очікується через його новизну та інтерес цільової аудиторії до інноваційних рішень
3.	Фактор якості послуг	Користувачі потребують метод, який працює для різних вхідних даних

За визначеними у таблиці 6.10 факторами конкурентоспроможності у таблиці 6.11 проведено аналіз сильних та слабких сторін стартап-проекту.

Таблиця 6.11 – Порівняльний аналіз сильних та слабких сторін системи кластеризації помилок автотестування

№ п/п	Фактор конкурентоспроможності	Бали 1-20	Рейтинг товарів-конкурентів у порівнянні з ReportPortal						
			-3	-2	-1	0	1	2	3
1.	Фактор часу	15				+			
2.	Фактор новизни товару	15			+				
3.	Фактор якості послуг	20	+						

Як фінальний етап ринкового аналізу можливостей впровадження є SWOT-аналіз на основі виділених ринкових загроз та можливостей, сильних та слабких сторін системи (таблиця 6.12). Перелік ринкових загроз та ринкових можливостей складається на основі аналізу факторів загроз та факторів можливостей маркетингового середовища. Ринкові загрози та ринкові можливості є наслідками (прогнозованими результатами) впливу факторів, і, на відміну від них, ще не є реалізованими на ринку та мають певну ймовірність здійснення.

Таблиця 6.12 – SWOT-аналіз стартап-проекту

Сильні сторони: Якість послуг, що надаються Новизна послуг Можливість використання сторонніх інвестицій та робочої сили	Слабкі сторони: Висока конкуренція
Можливості: Створення нової ринкової ніші Необхідність закладати у бюджет можливі ризики та зміни ринкових умов	Загрози: Різка зміна ринку систем аналізу помилок у сфері тестування

На основі SWOT-аналізу у таблиці 6.13 розроблені альтернативи ринкової поведінки для виведення стартап-проекту на ринок та орієнтовний оптимальний час їх ринкової реалізації з огляду на потенційні проекти конкурентів, що можуть бути виведені на ринок. Визначені альтернативи проаналізовано з точки зору строків та ймовірності отримання ресурсів.

Таблиця 6.13 – Альтернативи ринкового впровадження стартап-проекту

1	2	3	4
№ п/п	Альтернатива (орієнтовний комплекс заходів) ринкової поведінки	Ймовірність отримання ресурсів	Строки реалізації
1.	Ціль: отримання прибутку в короткостроковій перспективі. Конкуренція: цінова та партнерська (пропонуємо свої нові послуги розповсюдження інформації про магазини партнерів – рекламні послуги) Взаємодія з фірмами: активна боротьба за долю ринку, що належить конкурентам.	В короткостроковому плані – велика В довгостроковому плані – значний ризик втратити долю ринку, якщо займатися лише ціновою конкуренцією.	8 місяців-рік після запуску проекту

Кінець таблиці 6.13

1	2	3	4
2.	Ціль: захоплення частини ринку, підтримання її розміру та поступове нарощення об’ємів Конкуренція: нецінова (акцент на тому, що пропонуємо інноваційні послуги) Взаємодія з конкурентами: співпраця, активний моніторинг їх діяльності, при можливій появі реальних конкурентів можна запропонувати злиття компаній/проектів	Висока імовірність отримання ресурсів та утримання їх протягом довгого проміжку часу. Більш імовірний розвиток компаніїї та постійне покращення продукту.	8 місяців-рік після запуску проекту – для отримання перших фінансових надходжень від розповсюдження інформації про акції магазинів-партнерів, та їх реклама. Далі фінансові надходження прогнозовано регулярними.

Таким чином можна побачити, що альтернатива номер 1 передбачає більш просте та ймовірне отримання ресурсів, а також більш стислі строки реалізації, тому обираємо її.

6.4 Розроблення ринкової стратегії проекту

Розроблення ринкової стратегії першим кроком передбачає визначення стратегії охоплення ринку. У таблиці 6.14 наведено опис цільових груп потенційних споживачів.

Таблиця 6.14 – Вибір цільових груп потенційних споживачів

1	2	3	4	5	6
№ п/п	Опис профілю цільової групи потенційних клієнтів	Готовність споживачів сприйняти продукт	Орієнтовний попит в межах цільової групи (сегменту)	Інтенсивність конкуренції в сегменті	Простота входу у сегмент
1.	Високозабезпечені люди, які зацікавлені у пошуку перспективних проектів для інвестування	Споживачі слідкують з найновітніши ми технологіями, бажають бути в тренді та готові сприйняти новий продукт	Потенційно високий, інвестори хочуть бути впевненими у доцільності своїх інвестицій та подальшому отримання прибутку	Практично відсутня	Досить просто

Кінець таблиці 6.14

1	2	3	4	5	6
2.	Ініціативні люди, які мають хорошу ідею для розробки системи побудови фоторобота, та хочуть втілити її у життя	Споживачі готові сприйняти продукт, так як зацікавлені у глибинному аналізі ситуації	Високий попит	Прак-тично від-сутня	Досить просто

Таким чином за результатами аналізу потенційних груп споживачів було обрано першу та другу цільову групу, якій може бути запропонований розроблюваний програмний продукт, та обрана стратегія охоплення ринку – концентрований маркетинг.

Для роботи в обраному сегменті ринку у таблиці 6.15 сформована базова стратегія розвитку.

Таблиця 6.15 – Визначення базової стратегії розвитку

№ п/п	Обрана альтернатива розвитку проекту	Стратегія охоплення ринку	Ключові конкурентоспроможні позиції відповідно до обраної альтернативи	Базова стратегія розвитку
1.	Захоплення, підтримання та захист частки ринку	Стратегія концентрованого маркетингу	Новизна послуг Доступність продукту Простота в користуванні продуктом Додаткові зручні аспекти, які враховуються для системи побудови фоторобота, що вигідно виділяють наш продукт серед конкурентів	Стратегія диферен-ціації

Наступним кроком у таблиці 6.16 обрано стратегію конкурентної поведінки.

Таблиця 6.16. Визначення базової стратегії конкурентної поведінки

№ п/п	Чи є проект першопрохідцем на ринку?	Чи буде компанія шукати нових споживачів або забирати існуючих у конкурентів?	Чи буде компанія копіювати основні характеристики товару конкурента, і які?	Стратегія конкурентної поведінки
1.	Ні	І те, і те	Частково, загальні засади роботи продукту	Стратегія лідера

На основі вимог споживачів з обраного сегменту до стартап-компанії та до продукту, а також в залежності від обраної у таблиці 6.15 базової стратегії розвитку й обраної у таблиці 6.16 базової стратегії конкурентної поведінки розроблена і наведена у таблиці 6.17 стратегія позиціонування, що полягає у формування ринкової позиції, тобто комплексу асоціацій, за якими споживачі мають ідентифікувати проект.

Таблиця 6.17 – Визначення стратегії позиціонування

1	2	3	4	5
№ п/п	Вимоги до товару цільової аудиторії	Базова стратегія розвитку	Ключові конкурентоспроможні позиції власного стартап-проекту	Вибір асоціацій, які мають сформувати комплексну позицію власного проекту (три ключових)
1.	Універсальність методу оцінювання з точки зору інвестора	Стратегія диференціації	Врахування всіх аспектів оцінювання проекту з точки зору інвестиційної привабливості	Ваші гроші ефективно працюють у інноваційному прогресивному проекті
2.	Універсальність методу оцінювання з точки зору команди проекту	Стратегія диференціації	Врахування всіх аспектів оцінювання проекту з точки зору інвестиційної привабливості та життєздатності проекту, чи доцільно інноваційних проект реалізовувати	Реальна можливість втілити у життя інноваційний проект завдяки глибинному аналізу ключових аспектів та пошуку інвесторів

Кінець таблиці 6.17

3.	Необхідність враховувати ризики проекту, ринкові та економічні умови, що швидко змінюються	Стратегія диференціації	Врахування ключових ризиків та ринкових умов завдяки розробленій системі коефіцієнтів	Детальний облік ризиків та моніторинг ринкових умов дозволять уникнути передчасного закриття проекту

Таким чином як результат цього підрозділу ми отримали узгоджену систему рішень щодо ринкової поведінки стартап-компанії, яка визначатиме напрями роботи стартап-компанії на ринку.

6.5 Розроблення маркетингової програми стартап-проекту

Першим кроком є формування маркетингової концепції товару, який отримає споживач. У таблиці 6.18 підсумовано результати попереднього аналізу конкурентоспроможності товару.

Таблиця 6.18 – Визначення ключових переваг концепції потенційного товару

№ п/п	Потреба	Вигода, яку пропонує товар	Ключові переваги перед конкурентами (існуючі або такі, що потрібно створити)
1.	Універсальна інтелектуальна система кластеризації помилок тестування, яка буде корисною як для інвесторів, так і для команди	Спрощення та прискорення процесу аналізу результатів тестування та аналізу причин неуспішності тестів	Реалізація алгоритму кластеризації помилок, що забезпечує виявлення закономірностей у вхідних даних, кластерів невідомих помилок

У наступній таблиці (6.19) розроблена трирівнева маркетингова модель товару.

Таблиця 6.19 – Опис трьох рівнів моделі товару

1	2
Рівні товару	Сутність та складові
I. Товар за задумом	Опис: інтелектуальний метод аналізу результатів тестування, що виявляє закономірності у помилках та може бути використаний командами тестувальників малих та великих компаній

Кінець таблиці 6.19

II. Товар у реальному виконанні	Властивості/характеристики
	Універсальна система кластеризації повідомлень про помилку у виконанні тесту
	Пакування: програмна бібліотека
III. Товар із підкріпленням	До продажу
	Після продажу
	За рахунок чого потенційний товар буде захищено від копіювання: захист інтелектуальної власності розробленої системи кластеризації помилок (ліцензія)

Наступним кроком є визначення цінових меж, якими необхідно керуватись при встановленні ціни на потенційний товар, яке передбачає аналіз ціни на товари-аналоги або товари субститути, а також аналіз рівня доходів цільової групи споживачів. Аналіз експертним методом проведено у таблиці 6.20.

Таблиця 6.20 – Визначення меж встановлення ціни

№ п/п	Рівень цін на товари-замінники	Рівень цін на товари-аналоги	Рівень доходів цільової групи споживачів	Верхня та нижня межі встановлення ціни на товар
1.	Безкоштовно	Безкоштовно	Більше 10000 грн./місяць	-

Наступним кроком є визначення оптимальної системи збуту, в межах якого приймається рішення, що описано у таблиці 6.21.

Таблиця 6.21 – Формування системи збуту

№ п/п	Специфіка закупівельної поведінки цільових клієнтів	Функції збуту, які має виконувати постачальник товару	Глибина каналу збуту	Оптимальна система збуту
1.	Купують право на використання алгоритму	Зберігання, сортування, встановлення контакту, інформування	Однорівневий	Залучена

Останньою складовою маркетингової програми є розроблення концепції маркетингових комунікацій, що спирається на попередньо обрану основу для позиціонування, визначену специфіку поведінки клієнтів і проілюстровано таблицею 6.22.

Таблиця 6.22 – Концепція маркетингових комунікацій

№ п/п	Специфіка поведінки цільових клієнтів	Канали комунікації, якими користуються цільові клієнти	Ключові позиції, обрані для позиціонування	Завдання рекламного повідомлення	Концепція рекламного повідомлення
1.	Розвиток технологій, активність розробки програмного забезпечення, важливість та актуальність проведення тестування створюють потребу у застосуванні інтелектуальної системи аналізу результатів тестування	Соціальні мережі, внутрішні комунікаційні канали IT-компаній	Система кластеризації помилок як складова автоматизації тестування	Впевнити клієнта у тому, що даний метож є універсальним та оптимальним для вирішення його задач	Повідомлення у соціальних мережах, короткі ролики

Результатом даного підрозділу є маркетингова програма, що включає в себе концепції товару, збуту, просування та попередній аналіз можливостей ціноутворення, спирається на цінності та потреби потенційних клієнтів, конкурентні переваги ідеї, стан та динаміку

ринкового середовища, в межах якого буде впроваджено проект, та відповідну обрану альтернативу ринкової поведінки.

Висновок до розділу 6

У даному розділі проведено аналіз ідеї та маркетингової стратегії стартап-проекту з розробки системи кластеризації даних для застосування в інтелектуальному аналізі результатів тестування.

Можливість ринкової комерціалізації присутня, наявний попит, ринок потенційно відкритий для інновацій, а рентабельність вища за прибутковість банківських вкладів.

Перспективи впровадження з огляду на потенційні групи клієнтів наявні, ринок перспективний. Конкуренції майже немає при високій конкурентоспроможності даного продукту.

Для ринкової реалізації продукту доцільно обрати альтернативу, направлену на довгострокову роботу та утримання клієнтів, роботу над покращенням розробленого методу кластеризації помилок із використанням альтернативних метрик та нових технологій.

Подальша імплементація проекту доцільна, якщо інвестори готові до того, що проект окупиться за майже три роки.

ВИСНОВКИ

У даній магістерській дисертації проаналізовано задачу кластеризації помилок тестування. Встановлено, що більшість наявних засобів для агрегації результатів тестування, наприклад, SerilogAmazonKinesis, AventStack.ExtentReports, Specrunner, не забезпечують інтелектуального аналізу, а наявна система інтелектуального аналізу помилок тестування ReportPortal є неточною та, за відгуками, видає неактуальний результат. Проаналізовано особливості предметної області та вирішено, що алгоритми кластеризації зможуть забезпечити якісний аналіз та виявлення закономірностей у повідомленнях про помилки. Сформульовано постановку задачі магістерської дисертації, виділено етапи її розвитку, введено обмеження роботи.

Досліджено такі методи векторизації текстових повідомлень, як Bag of words, TF-TDF, Word2Vec. Виявлено їхні переваги та недоліки. Хоч метод Word2Vec дуже потужний та має високу якість роботи, проте пошук семантичних зв’язків між словами є неактуальним для розв’язку поставленої задачі, тому даний алгоритм визнано не застосовним у даній магістерській дисертації. Як найкращий для розв’язку задачі векторизації повідомлень про помилки у тестуванні обрано алгоритм TF-IDF із модифікацією словника.

Досліджено моделі кластеризації даних, а саме нейронні, центроїдні, статистичні, ієрархічні, щільнісні та графові. Введено критерії якості алгоритмів та проаналізовано такі алгоритми кластеризації, як нейронні мережі Кохонена, агломеративний підхід, EM-алгоритм, DBSCAN, HCS. Виявлено їхні переваги та недоліки. Обрано найкращий, за визначеними критеріями, алгоритм для розв’язку

задачі даної магістерської дисертації – агломеративний. Даний алгоритм не потребує апріорних знань про дані, добре працює із викидами, не залежить від ініціалізації та має помірну обчислювальну складність. Розглянуто типи відстаней, зв'язків та критеріїв оцінювання якості кластерного розбиття, визначено набір метрик для модифікації обраного алгоритму.

Проаналізовано вимоги користувача до функціоналу системи кластеризації помилок тестування. На основі проведеного до цього аналізу, визначено архітектуру системи, яка складається із модулів десеріалізації, веторизації, кластеризації, оцінки розбиття та логування; наведено діаграми класів її модулів. Алгоритм реалізовано засобами мови C# у вигляді класової бібліотеки.

Реалізовану систему протестовано на трьох наборах реальних даних, що містять 80, 150 і 300 записів про помилки тестування. Визначено, що для характеру обраної предметної області найкраще підходить неевклідова відстань міських кварталів. Робота алгоритму практично не залежить від обраного типу зв'язку, а час виконання напряму залежить від обсягу вхідних даних. Евристично визначено, що такі критерії оцінювання якості кластеризації, як коефіцієнт середнього силуету, індекс Данна та within-between метрика, видають коректні результати лише для відносно не великої кількості повідомлень (до 300).

Загалом реалізований алгоритм працює коректно та реалізовує поставлену у даній магістерській дисертації задачу.

Для подальшого дослідження рекомендовано розглянути інші методи визначення якості розбиття даних на кластери, виявити та реалізувати застосовні до великого набору помилок. Також запропоновано на основі розробленої бібліотеки реалізувати REST сервіс з інтуїтивно зрозумілим API для запуску системи кластеризації помилок.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Malashniak M. Software development in Ukraine: 2019-2020 IT market report. *N-ix*. August 29, 2019. URL: https://www.n-ix.com/software-development-in-ukraine-2019-2020-market-report/

2. Indira R. Test Result Reporting. *Infosys*: 3rd Annual International Software Testing Conference materials. 2001. URL: https://www.infosys.com/IT-services/validation-solutions/white-papers/Documents/test-result-reporting.pdf.

3. Pantola P. Natural Language Processing: Text Data Vectorisation. *Medium*. 2018. URL: https://medium.com/@paritosh_30025/natural-language-processing-text-data-vectorization-af2520529cf7.

4. Mayo M. Text Data Preprocessing: A Walkthrouhg in Python. *KDnuggets*. 2018. URL: https://www.kdnuggets.com/2018/03/text-data-preprocessing-walkthrough-python.html.

5. Ojeda T., Bilbro R., Bengford B. Applied Text Analysis with Python. O'Reilly Media, Inc., 2018. ISBN 9781491963036. URL: https://www.oreilly.com/library/view/applied-text-analysis/9781491963036/ch04.html.

6. Brownlee J. A Gentle Introduction to the Bag-of-Words Model. *Machine Learning Mastery*. 2017. URL: https://machinelearningmastery.com/gentle-introduction-bag-words-model/.

7. Critchlow W. A Beginner's Guide to word2vec AKA What's the Opposite of Canada? *Distilled*. 2016. URL: https://www.distilled.net/word2vec-examples/.

8. Навчання без учителя. *Вікіпедія*. URL: https://uk.wikipedia.org/ wiki/Навчання_без_учителя.

9. Начання без вчителя. *Знаймо*. URL: http://znaimo.com.ua/Навчання_без_вчителя.

10. Кластерний аналіз. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/ Кластерний_аналіз.

11. Штучна нейронна мережа. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Штучна_нейронна_мережа.

12. Нейронна мережа Кохонена. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Нейронна_мережа_Кохонена.

13. Sarle W. S. How many finds of Kohonen networks exist? *comp.ai.neural-nets*. URL: http://www.faqs.org/faqs/ai-faq/neural-nets/part1/section-11.html.

14. Hecht-Nielsen R. Neurocomputing. MA: Addison-Wesley, 1990. 560 pp. ISBN 0-201-09355-3.

15. Kohonen T. Learning Vector Quantization: Neural Networks. 1988. 988 pp.

16. Кластеризація методом к-середніх. *Вікіпедія*. URL: https://uk.wikipedia.org/wiki/Кластеризація_методом_к–середніх.

17. Hierarchical clustering. *ALGLIB User Guide*. URL: http://www.alglib.net/dataanalysis/clustering.php.

18. Kaufman L., Rousseeuw P. J. Finding Groups in Data. An Introduction to Cluster Analysis. US: Wiley-Interscience, A John Wiley & Sons, Inc., 2005. 763 pp. ISBN 0-47 1-73578-7.

19. Sequeira P. Aglomera.NET. *GitHub*. URL: https://github.com/pedrodbs/Aglomera

20. Camargos R. C. Finding groups in data with C# – Agglomerative Clustering. *Code Project.* 2016. URL: https://www.codeproject.com/Articles/1120804/Finding-groups-in-data-with-Csharp-Agglomerative-C

21. Campello R., Moulavi D., Sander J. Density-Based Clustering Based on Hierarchical Density Estimates. *Advances in Knowledge Discovery and Data Mining*. Springer. 2013. Pp 160-172.

22. Doxakis P. HdbscanSharp. *GitHub*. URL: https://github.com/doxakis/HdbscanSharp

23. McInnes L, Healy J. Accelerated Hierarchical Density Based Clustering. 2017 IEEE International Conference on Data Mining Workshops (ICDMW). IEEE. 2017. Pp 33-42. URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8215642

24. Eldridge J., Belkin M., Wang Y. Beyound Hartigan Consistency: Merge Distortion Metric for Hierarchical Clustering. *arXiv*. 2015. URL: https://arxiv.org/pdf/1506.06422v2.pdf

25. How HDBSCAN Works. *Jupiter nbviewer*. URL: https://nbviewer.jupyter.org/github/scikit-learn-contrib/hdbscan/blob/master/notebooks/How%20HDBSCAN%20Works.ipynb

26. HCS clustering algorithm. *Wikipedia*. URL: https://en.wikipedia.org/wiki/HCS_clustering_algorithm

27. Kassambara A. Cluster Validation Statistics: Must Know Methods. CLUSTER VALIDATION ESSENTIALS. *DATA NOVIA*. URL: https://www.datanovia.com/en/lessons/cluster-validation-statistics-must-know-methods/#internal-measures-for-cluster-validation

28. Dunn J. C. A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics.* Volume 3, 1973. Pp. 32-57. URL: https://doi.org/10.1080/01969727308546046.

29. Zhao Q., Xu M., Fränti P. Sum-of-Squares Based Cluster Validity Index and Significance Analysis. *Adaptive and Natural Computing Algorithms*. Springer. Volume 5495, 2009. Pp. 313-322.

ДОДАТОК А

Лістинг коду програми

ResultSummary.cs

```
using
System.Xml.Serialization;

namespace
Deserialization.Models.Xml
{
    public class ResultSummary
    {

[XmlAttribute("outcome")]
        public string Outcome
{ get; set; }

        public Counters
Counters { get; set; }
    }

    public class Counters
    {

[XmlAttribute("total")]
        public string Total {
get; set; }


[XmlAttribute("executed")]
        public string Executed
{ get; set; }


[XmlAttribute("passed")]
        public string Passed {
get; set; }


[XmlAttribute("failed")]
        public string Failed {
get; set; }


[XmlAttribute("inconclusive")]
        public string
Inconclusive { get; set; }


[XmlAttribute("notExecuted")]
        public string
NotExecuted { get; set; }
    }
}
```

TestRun.cs

```
using System.Xml;
using System.Xml.Serialization;

namespace Deserialization.Models.Xml
{
    [XmlRoot(Namespace =
"http://microsoft.com/schemas/Visual
Studio/TeamTest/2010", IsNullable =
false)]
    public class TestRun
    {
        [XmlAnyElement]
        XmlNode Attribute { get;
set; }

        [XmlAttribute("id")]
        public string Id { get; set;
}

        [XmlArray]
        public UnitTestResult[]
Results { get; set; }

        [XmlArray]
        public UnitTest[]
TestDefinitions { get; set; }

        public ResultSummary
ResultSummary { get; set; }
    }
}
```

UnitTest.cs

```
using System.Xml.Serialization;

namespace Deserialization.Models.Xml
{
    public class UnitTest
    {
        [XmlAttribute("id")]
        public string Id { get; set;
}
```

```
        [XmlArray(IsNullable = true)]
        public TestCategoryItem[] TestCategory { get; set; }

        public TestMethod TestMethod { get; set; }
    }

    public class TestCategoryItem
    {
        [XmlAttribute]
        public string TestCategory { get; set; }
    }

    public class TestMethod
    {
        [XmlAttribute("name")]
        public string Name { get; set; }

        [XmlAttribute("className")]
        public string ClassName { get; set; }
    }
}
```

UnitTestResult.cs

```
using System.Xml.Serialization;

namespace Deserialization.Models.Xml
{
    public class UnitTestResult
    {
        [XmlAttribute("testId")]
        public string TestId { get; set; }

        [XmlAttribute("testName")]
        public string TestName { get; set; }

        [XmlAttribute("duration")]
        public string Duration { get; set; }

        [XmlAttribute("outcome")]
        public string Outcome { get; set; }

        public Output Output { get; set; }
    }

    public class Output
    {
        public string StdOut { get; set; }

        [XmlElement(IsNullable = true)]
        public ErrorInfo ErrorInfo { get; set; }
    }

    public class ErrorInfo
    {
        public string Message { get; set; }

        public string StackTrace { get; set; }
    }
}
```

TestResult.cs

```
using Deserialization.Models.Xml;

namespace Deserialization.Models
{
    public class TestResult
    {
        public string TestId { get; set; }
        public string TestName { get; set; }
        public string Feature { get; set; }
        public string Duration { get; set; }
        public string Outcome { get; set; }
        public string Message { get; set; }
        public Output FullOutput { get; set; }
    }
}
```

Deserializer.cs

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml;
using System.Xml.Serialization;
using Deserialization.Constants;
using Deserialization.Models;
using Deserialization.Models.Xml;
using Logging.Logger;

namespace Deserialization
```

```
{
    public static class Deserializer
    {
        public     static     TestRun
Deserialize(string fileName)
        {
            XmlSerializer  serializer
=                                  new
XmlSerializer(typeof(TestRun));
            serializer.UnknownNode
+= serializer_UnknownNode;

serializer.UnknownAttribute       +=
serializer_UnknownAttribute;
            FileStream  fileStream  =
new              FileStream(fileName,
FileMode.Open);

            return
(TestRun)serializer.Deserialize(file
Stream);
        }

        public                  static
IEnumerable<TestResult>
GetFailedTestsResults(this    TestRun
testRun)
        {
            var tempTestsNamesList =
new List<string>();
            return
testRun.Results.Select(result =>
            {
                var    testName    =
result.TestName;

tempTestsNamesList.Add(testName);

                return            new
TestResult
                {
                    TestId          =
result.TestId,
                    TestName        =
testName + "_run"

+ tempTestsNamesList.FindAll(item =>
item.Equals(testName)).Count,
                    Feature         =
testRun.TestDefinitions.Single(test
=> test.Id.Equals(result.TestId))

.TestMethod.ClassName.Split('.').Las
t(),
                    Duration        =
result.Duration,
                    Outcome         =
result.Outcome,
                    Message         =
result.Output?.ErrorInfo?.Message,
                    FullOutput      =
result.Output
                };
            }).Where(result       =>
result.Outcome.Equals(nameof(Outcome
s.Failed)));
        }

        private      static      void
serializer_UnknownNode(object
sender, XmlNodeEventArgs e)
        {
//                  Log.Message("Non-
deserialized Node:" + e.Name + "\t"
+ e.Text);
        }

        private      static      void
serializer_UnknownAttribute(object
sender, XmlAttributeEventArgs e)
        {
            XmlAttribute    attr    =
e.Attr;
//                Log.Message($"Non-
deserialized attribute {attr.Name} =
'{attr.Value}'");
        }
    }
}
```

Outcomes.cs

```
namespace Deserialization.Constants
{
    public enum Outcomes
    {
        Passed,
        Failed
    }
}
```

ILog.cs

```
namespace Logging.Logger
{
    public interface ILog
    {
        void  Message(LogMessageTypes
types,    string    message,    params
string[] param);

        void Comment(string message,
params string[] param);
```

```
        void          Message(string
message);

        void  Trace(string  message,
params string[] param);
    }
}
```

FileLog.cs

```
using System;
using System.IO;

namespace Logging.Logger
{
    public class FileLog : ILog
    {
        StreamWriter  Stream  {  get;
set; }

        public   FileLog(StreamWriter
stream) => Stream = stream;

        public                   void
Message(LogMessageTypes       types,
string   message,   params   string[]
param)
        {
            string indent = "";
            switch (types)
            {
                case
LogMessageTypes.Comment:
                {
                    message        =
"Comment: " + message;
                    break;
                }
                case
LogMessageTypes.Message:
                {
                    indent = "\t";
                    break;
                }
                case
LogMessageTypes.Trace:
                {
                    indent = "\t\t";
                    break;
                }
            }


//                        message  =
string.Format(message, param);
            string      time      =
string.Format("({0})",
DateTime.Now.ToLongTimeString());
            string  newLineIndent  =
new string(' ', 14);
            message   =   indent   +
message.Replace("\n",     "\n"     +
newLineIndent + indent);
            message                =
string.Format("{0,-14}{1}",    time,
message);

this.Stream.WriteLine(message);
        }

        public  void  Comment(string
message, params string[] param)
        {

this.Message(LogMessageTypes.Comment
, message, param);
        }

        public  void  Message(string
message)
        {

this.Message(LogMessageTypes.Message
, message);
        }

        public   void   Trace(string
message, params string[] param)
        {

this.Message(LogMessageTypes.Trace,
message, param);
        }
    }
}
```

ConsoleLog.cs

```
using System;

namespace Logging.Logger
{
    public class ConsoleLog : ILog
    {
        public                   void
Message(LogMessageTypes       types,
string   message,   params   string[]
param)
        {
            string indent = "";
            switch (types)
            {
                case
LogMessageTypes.Comment:
                {
```

```
                    message         =
"Comment: " + message;
                    break;
                }
                case
LogMessageTypes.Message:
                {
                    indent = "\t";
                    break;
                }
                case
LogMessageTypes.Trace:
                {
                    indent = "\t\t";
                    break;
                }
            }

//                          message  =
string.Format(message, param);
            string       time       =
string.Format("({0})",
DateTime.Now.ToLongTimeString());
            string  newLineIndent  =
new string(' ', 14);
            message   =   indent   +
message.Replace("\n",     "\n"     +
newLineIndent + indent);
            message                 =
string.Format("{0,-14}{1}",     time,
message);

Console.WriteLine(message);
        }

        public  void  Comment(string
message, params string[] param)
        {

this.Message(LogMessageTypes.Comment
, message, param);
        }

        public  void  Message(string
message)
        {

this.Message(LogMessageTypes.Message
, message);
        }

        public   void   Trace(string
message, params string[] param)
        {

this.Message(LogMessageTypes.Trace,
message, param);
        }
    }
}
```

LogMessageTypes.cs

```
namespace Logging
{
    public enum LogMessageTypes
    {
        Comment,
        Message,
        Trace
    }
}
```

Vectorizer.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using Deserialization.Models;
using Logging.Logger;
using TfidfDotNet;

namespace Text.TfIdf
{
    public static class Vectorizer
    {
        internal               static
TfidfVocabulary  Vocabulary  {  get;
set; }

        internal static List<string>
GetVocabulary()
        {
            var   termsList   =   new
List<string>();
            foreach   (var   s   in
Vocabulary) termsList.Add(s);
            return termsList;
        }

        public   static   double[][]
Vectorize(List<string> results)
        {
            Vocabulary             =
TfidfVectorizer.GenerateVocabulary(r
esults,
                out var frequencies,
                new
TfidfSettings());

            var   tempVocabulary   =
Vocabulary;
            return
TfidfVectorizer.Transform(results,
ref tempVocabulary);
        }
```

```
        public                static
KeyValuePair<string,      double>[][]
GetDescription(this        double[][]
vectorized)
        {
            var   doubleArrayList   =
vectorized.ToList();
            return
doubleArrayList.Select(doubleArray
=>
            {
                var    doubleList    =
doubleArray.ToList();
                var    termsList    =
GetVocabulary();
                var    termsKeys    =
doubleList
                    .Where(i       =>
doubleList.IndexOf(i) % 2 == 0)
                    .Select(i      =>
(int)i)
                    .Select(i      =>
termsList[i - 1])
                    .ToList();
                var     values     =
doubleList
                    .Where(i       =>
doubleList.IndexOf(i) % 2 != 0)
                    .ToList();
                var    log    =    new
ConsoleLog();
                log.Message("\nNext
message:");
                return
termsKeys.Select(key => {
                    var    value    =
values[termsKeys.IndexOf(key)];

log.Message($"{key}: {value}");
                    return        new
KeyValuePair<string,     double>(key,
value);
                }).ToArray();
            }).ToArray();
        }

        public  static  List<(string,
double[])>
VectorizeTestResultsAndReturnForData
Point(List<TestResult> testResults)
        {
            var     vectorized     =
Vectorize(testResults.Select(result
=> result.Message)
                .Where(message     =>
message != null).ToList());
            var        terms        =
GetVocabulary();
            return
testResults.Select(testResult =>
            {
                var    valuesList    =
vectorized[testResults.IndexOf(testR
esult)].ToList();
                var   termsIndices   =
valuesList
                    .Where(i       =>
valuesList.IndexOf(i) % 2 == 0)
                    .Select(i      =>
(int) i);
                var   fullTermsValues
= terms.Select(term =>
                {
                    var   indexOfTerm
= terms.IndexOf(term);
                    return
termsIndices.Contains(indexOfTerm)
                        ?
valuesList[valuesList.IndexOf(indexO
fTerm) + 1]
                        : 0;
                });
                return            new
ValueTuple<string,
double[]>(testResult.TestName,
fullTermsValues.ToArray());
            }).ToList();
        }
    }
}
```

DataPoint.cs

```
using System;
using System.Linq;
using System.Text;
using Aglomera;

namespace
Clustering.Hierarchical.Models
{
    class         DataPoint         :
IEquatable<DataPoint>,
IDissimilarityMetric<DataPoint>,
IComparable<DataPoint>
    {


        #region     Properties     &
Indexers

        public   string   ID   {   get;
protected set; }

        public double[] Value { get;
protected set; }
```

```
        #endregion

        #region Public Methods

        public override bool Equals(object obj) => obj is DataPoint && this.Equals((DataPoint)obj);

        public override int GetHashCode() => this.ID.GetHashCode();

        public override string ToString() => this.ID;

        #endregion

        #region Public Methods

        public virtual double DistanceTo(DataPoint other)
        {
            throw new NotImplementedException();
        }

        public int CompareTo(DataPoint other) => string.Compare(this.ID, other.ID, StringComparison.Ordinal);

        public double Calculate(DataPoint instance1, DataPoint instance2) => instance1.DistanceTo(instance2);

        public bool Equals(DataPoint other) => string.Equals(this.ID, other.ID);

        #endregion
    }
}
```

EuclideanDataPoint.cs

```
using System;
using System.Linq;
using System.Text;
using Aglomera;

namespace Clustering.Hierarchical.Models
{
    class EuclideanDataPoint : DataPoint
    {

        public EuclideanDataPoint(string id, double[] value)
        {
            this.ID = id;
            this.Value = value;
        }

        public EuclideanDataPoint()
        {
        }

        public override bool Equals(object obj) => obj is EuclideanDataPoint && this.Equals((EuclideanDataPoint)obj);

        public override int GetHashCode() => this.ID.GetHashCode();

        #region Public Methods

        public EuclideanDataPoint GetCentroid<T>(Cluster<T> cluster) where T: DataPoint
        {
            if (cluster.Count == 1) return cluster.First() as EuclideanDataPoint;

            // gets sum for all variables
            var id = new StringBuilder();
            var sums = new double[cluster.First().Value.Length];
            foreach (var dataPoint in cluster)
            {

                id.Append(dataPoint.ID);
                for (var i = 0; i < sums.Length; i++)
                    sums[i] += dataPoint.Value[i];
            }

            // gets average of all variables (centroid)
            for (var i = 0; i < sums.Length; i++)
                sums[i] /= cluster.Count;
```

```
            return                new
EuclideanDataPoint(id.ToString(),
sums);
        }

        public                static
EuclideanDataPoint
GetMedoid(Cluster<EuclideanDataPoint
> cluster) => cluster.GetMedoid(new
EuclideanDataPoint());

        public static bool operator
==(EuclideanDataPoint          left,
EuclideanDataPoint     right)     =>
left.Equals(right);

        public static bool operator
!=(EuclideanDataPoint          left,
EuclideanDataPoint     right)     =>
!left.Equals(right);

        public                double
DistanceTo(EuclideanDataPoint other)
        {
            var sum2 = 0d;
            var       length       =
Math.Min(this.Value.Length,
other.Value.Length);
            for (var idx1 = 0; idx1
< length; ++idx1)
            {
                var      delta      =
this.Value[idx1]                    -
other.Value[idx1];
                sum2   +=   delta   *
delta;
            }

            return Math.Sqrt(sum2);
        }

        public    override    double
DistanceTo(DataPoint other)
        {
            return   DistanceTo(other
as EuclideanDataPoint);
        }

        #endregion
    }
}
```

ManhattanDataPoint.cs

```
using System;
using System.Linq;
using System.Text;
using Aglomera;

namespace
Clustering.Hierarchical.Models
{
    class    ManhattanDataPoint    :
DataPoint
    {
        #region Constructors

        public
ManhattanDataPoint(string        id,
double[] value)
        {
            this.ID = id;
            this.Value = value;
        }

        public ManhattanDataPoint()
        {
        }

        #endregion


        #region Public Methods

        public     override     bool
Equals(object   obj)   =>   obj   is
ManhattanDataPoint                 &&
this.Equals((ManhattanDataPoint)obj)
;

        public      override      int
GetHashCode()                       =>
this.ID.GetHashCode();

        public    override    string
ToString() => this.ID;

        #endregion

        #region Public Methods

        public    ManhattanDataPoint
GetCentroid<T>(Cluster<T>    cluster)
where T: DataPoint
        {
            if (cluster.Count == 1)
return       cluster.First()       as
ManhattanDataPoint;

            //  gets  sum  for  all
variables
            var     id     =     new
StringBuilder();
            var    sums    =    new
double[cluster.First().Value.Length]
;
```

```
            foreach (var dataPoint
in cluster)
            {

id.Append(dataPoint.ID);
                for (var i = 0; i <
sums.Length; i++)
                    sums[i]       +=
dataPoint.Value[i];
            }

            // gets average of all
variables (centroid)
            for (var i = 0; i <
sums.Length; i++)
                sums[i]           /=
cluster.Count;

            return                new
ManhattanDataPoint(id.ToString(),
sums);
        }

        public                static
ManhattanDataPoint
GetMedoid(Cluster<ManhattanDataPoint
> cluster) => cluster.GetMedoid(new
ManhattanDataPoint());

        public static bool operator
==(ManhattanDataPoint          left,
ManhattanDataPoint     right)     =>
left.Equals(right);

        public static bool operator
!=(ManhattanDataPoint          left,
ManhattanDataPoint     right)     =>
!left.Equals(right);

        public                double
DistanceTo(ManhattanDataPoint other)
        {
            var sum = 0d;
            var       length       =
Math.Min(this.Value.Length,
other.Value.Length);
            for (var idx1 = 0; idx1
< length; ++idx1)
            {
                var      delta      =
this.Value[idx1]                    -
other.Value[idx1];
                sum               +=
Math.Abs(delta);
            }

            return sum;
        }

        public    override    double
DistanceTo(DataPoint other)
        {
            return   DistanceTo(other
as ManhattanDataPoint);
        }

        public                    int
CompareTo(ManhattanDataPoint   other)
=> string.Compare(this.ID, other.ID,
StringComparison.Ordinal);

        public                double
Calculate(ManhattanDataPoint
instance1,         ManhattanDataPoint
instance2)                        =>
instance1.DistanceTo(instance2);

        public                   bool
Equals(ManhattanDataPoint  other)  =>
string.Equals(this.ID, other.ID);

        #endregion
    }
}
```

PerformanceMeasure.cs

```
using System;
using System.Diagnostics;

namespace
Clustering.Hierarchical.Utils
{
    class PerformanceMeasure
    {
        #region Fields

        private  readonly  Stopwatch
_timer = new Stopwatch();
        private long _memoryStart;

        #endregion

        #region Constructors

        public PerformanceMeasure()
        {
            this.TimeElapsed   =   new
TimeSpan();
        }

        #endregion

        #region     Properties     &
Indexers
```

```
        public long MemoryUsage { get; protected set; }

        public TimeSpan TimeElapsed { get; protected set; }

        #endregion

        #region Public methods

        public virtual void Start()
        {
            //starts measures (time and memory)
            GC.Collect();
            GC.WaitForPendingFinalizers();
            GC.Collect();
            this._memoryStart = Process.GetCurrentProcess().PrivateMemorySize64;
            this._timer.Start();
        }

        public virtual void Stop()
        {
            //stops timers and measures
            this._timer.Stop();
            var memoryEnd = Process.GetCurrentProcess().PrivateMemorySize64;

            this.TimeElapsed = this._timer.Elapsed;
            this.MemoryUsage += memoryEnd - this._memoryStart;
        }

        public void Reset()
        {
            // "zero"s all measures
            this._timer.Stop();
            this.MemoryUsage = 0;
            this.TimeElapsed = new TimeSpan();
        }

        public override string ToString()
        {
            return $"time elapsed: {this.TimeElapsed}, memory spent: {BytesToString(this.MemoryUsage)}";
        }

        /// <remarks>From <a href="http://stackoverflow.com/a/4975942" /></remarks>
        private static string BytesToString(long byteCount)
        {
            string[] suf = { "B", "KB", "MB", "GB", "TB", "PB", "EB" }; //Longs run out around EB
            if (byteCount == 0)
                return "0" + suf[0];
            var bytes = Math.Abs(byteCount);
            var place = Convert.ToInt32(Math.Floor(Math.Log(bytes, 1024)));
            var num = Math.Round(bytes / Math.Pow(1024, place), 1);
            return Math.Sign(byteCount) * num + suf[place];
        }

        #endregion
    }
}
```

JointDistanceFunction.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Aglomera;
using Aglomera.Evaluation.Internal;

namespace Clustering.Evaluation
{
    /// <summary>
    /// The joint distance function (JDF) J(x) at any point x is the harmonic mean of the distances
    /// between x and the cluster centers.
    /// </summary>
    /// <typeparam name="TInstance"></typeparam>
    public class JointDistanceFunction<TInstance> : IInternalEvaluationCriterion<TInstance> where TInstance : IComparable<TInstance>
    {
        public JointDistanceFunction(IDissimilarityMetric<TInstance> dissimilarityMetric, CentroidFunction<TInstance> centroidFunc)
```

```
        {                                                   }
            DissimilarityMetric     =           }
dissimilarityMetric;                        }
            CentroidFunction        =
centroidFunc;
        }

        public
IDissimilarityMetric<TInstance>
DissimilarityMetric { get; }
        public
CentroidFunction<TInstance>
CentroidFunction { get; }

        public                  double
Evaluate(ClusterSet<TInstance>
clusterSet)
        {
            if  (clusterSet.Count  <
2)
                return double.NaN;
            double      val1_1      =
double.MaxValue;
            double      val1_2      =
double.MinValue;
            for  (int  index1  =  0;
index1 < clusterSet.Count; ++index1)
            {
                List<TInstance>  list
=
clusterSet[index1].ToList<TInstance>
();
                for (int index2 = 0;
index2 < list.Count; ++index2)
                {
                    TInstance
instance1 = list[index2];
                    for  (int  index3
= index2 + 1; index3 < list.Count;
++index3)
                        val1_2      =
Math.Max(val1_2,
this.DissimilarityMetric.Calculate(i
nstance1, list[index3]));
                    for  (int  index3
=    index1    +    1;    index3    <
clusterSet.Count; ++index3)
                    {
                        foreach
(TInstance        instance2        in
clusterSet[index3])
                            val1_1  =
Math.Min(val1_1,
this.DissimilarityMetric.Calculate(i
nstance1, instance2));
                    }
                }
            }
            return val1_1 / val1_2;
```